6.046 10/4

Network Flow

Today: A lot of setup for problem

- Max Flow problem
- Cuts
- Augmenting Paths
- Ford Fulkerson Alg

---

Flow Network

directed graph $G(V,E)$

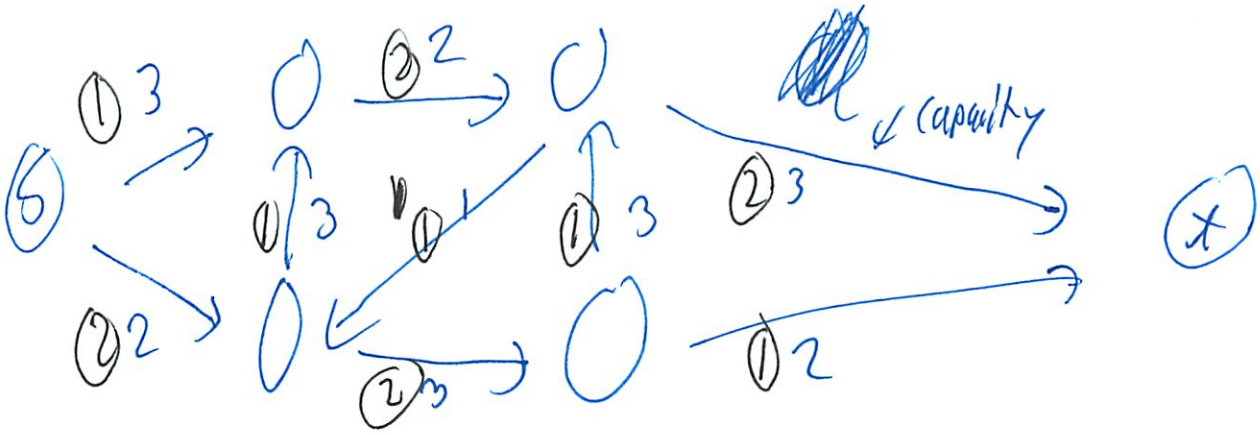
$s$ = Sorce
$t$ = Sink

each edge $(u,v) \in E$

$c(u,v) \geq 0$ ← capacity

if $0$ → don't put edge down
$\ominus$ → no notion of that today

if no edge $u \to v$ → no flow



Flow is seperate set of #s

in black

- diff than capacity
- must look an at overall network

Add more # to edges

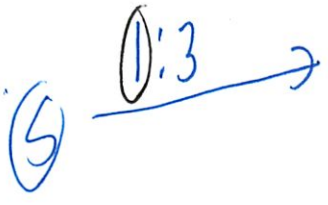
note: (constraints)

- flow ≤ capacity
- flow in = flow out
  except Sorce, Sink
  like Kircoffs Curent law
- flow out of sorce = flow into sink

③

We want to maximize possible flow of network

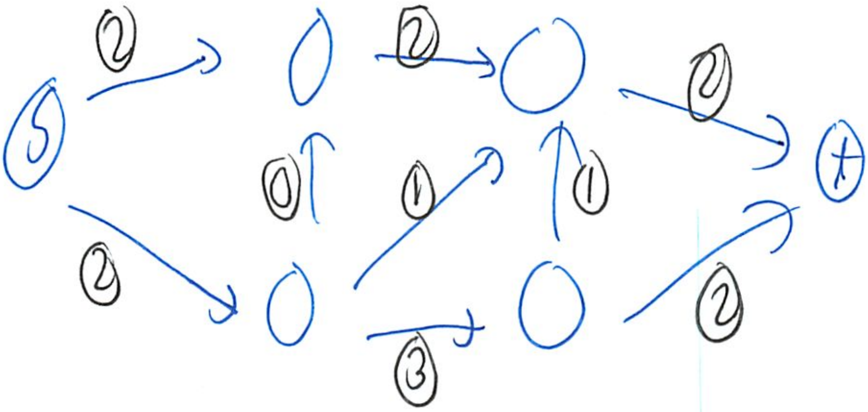
Could we change this around



to add more

But where would it go

But we can reconfigure whole network!



Is this a max flow?

~~[illegible] But how~~

How can you tell?

Could argue w/ individual edges

Can't reduce flow at first few nodes

But this gets messy!

(4)

We'll talk about a robust one late

Example of Applications

traffic

mobile phone ~~traff~~ tracking w/ Google Maps

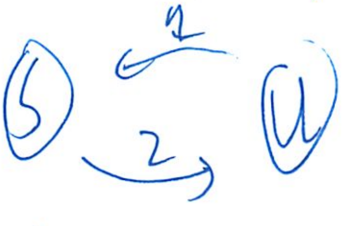
matching + dating

max flow alg

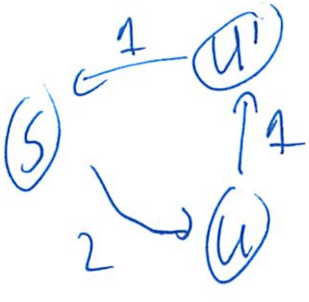
Assumptions

If edge $(u,v) \in E$ exists then $(v,u) \notin E$

No self loop edges exist



turn to dummy nodes



pos and net flow difference
so conect to 2nd graph & diff goes away

So talking about net flow

pos flow might be diff if we talk about this and don't have our simplification

## Net flow

Is a fn $f: V \times V \to R$

Satisfying - the capacity constraint

For all $u, v \in V$

$$f(u,v) \le c(u,v)$$

- flow conservation

For all $u \in V - \{s,t\}$

$$\sum_{v \in V} f(u,v) = 0$$

- skew symmetry

For all $u, v \in V$

$$f(u,v) = -f(v,u)$$

neg flow = other direction

Value of flow f, denoted $|f|$

$$|f| = \sum_{v \in V} f(s,v)$$

We want to maximize f, subject to our constraints

Implicit summation notation

$$= f(s,V)$$

no sigma

but set V here

So rewriting flow conservation

$$f(u,V) = 0 \text{ for all } u \in V - \{s,t\}$$

~~flow conservation~~

Simple properties

$$f(X,X) = 0$$

Say $X = \{x_1, x_2\}$

$$\underbrace{f(x_1,x_2) + f(x_2,x_1)}_{0} + f(\overset{0 \text{ defined as}}{\cancel{x_1,x_1}}) + f(\overset{0}{\cancel{x_2,x_2}})$$

$$+0 \quad 0 \quad +0$$

$$f(x, y) = -f(y, x)$$

opposite direction

$$f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$$

if $X \cap Y = \emptyset$

Theorm

$$|f| = f(V, t)$$

flow out of source = into sink

Proof $|f| = f(s, V)$ out flow from source

$$= \underset{0}{\cancel{f(V, V)}} - f(V - \{s\}, V)$$

$$= -f(V - s, V)$$

$$= f(V, V - s)$$

$$= f(V, t) + \underset{0}{\cancel{f(V,}}\ \underbrace{V - s - t}_{\text{disjoint}})$$

$= f(V, t)$ flow into sink

Prof: If confused → make up an example + play w/ it
"I do it"

---

## Cuts

Increment flow w/ augmented path

note: back to ~~original example~~
max flow example

Straightforward defn

Cut $(S,T)$

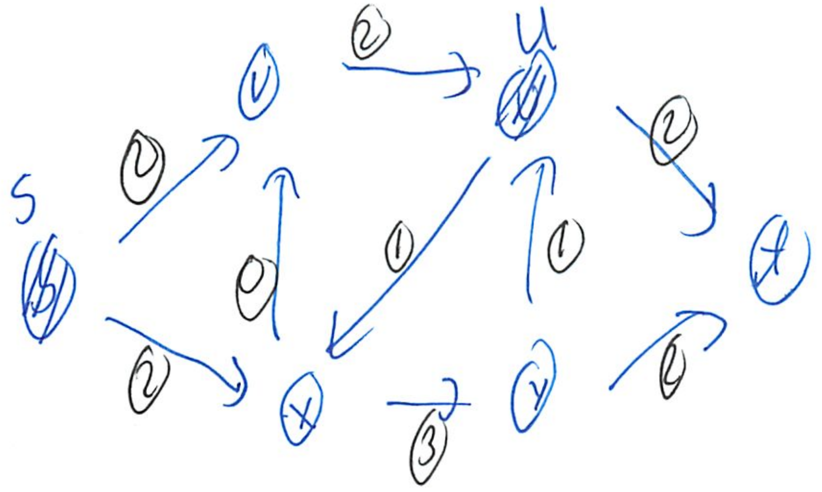
$S,T$ to be such that
$s \in S$ and $t \in T$

$S \cup T = V$ (partitioning)

more general that that
Can have arbitray nodes
as long as source is in $S$
sink $T$

NOT JUST a line over the network

9



$S = \{s, u\}$ shaded

$T =$ rest unshaded

$= \{v, x, y, t\}$

$f(S, T) =$

↑ sum up flows across all vertices such that have $S, T$

(missed detail)

$$= \underset{s,v}{2} + \underset{s,x}{2} + \underset{u,v}{-2} + \underset{u\,x}{1} + \underset{u\,y}{-1} + \underset{u\,t}{2}$$

↑ call the ones its selected to for each

$= 4$

↑ note same as max flow!

Lemma For any flow $f$ and any cut $(S,T)$ we have $|f| = f(S,T)$

? much more general

Proof w/ implicit summation looks similar to what we did

$$\begin{aligned} f(S,T) &= f(S,V) - \cancelto{0}{f(S,S)} \\ &= f(S,V) \\ &= f(s,V) + \underbrace{f(S-s,V)} \\ &= f(s,V) \\ &= |f| \end{aligned}$$

Zero ; yes
does not contain ~~t~~ t
since in T
or s since we subtracted it

flow through cut = flow source = flow sink

In the case where flow not max can we increase it?

Can compare arbitray cut to its capacity

Capacity of cut S,T is c(S,T)

$$c(S,T) = \underset{s,v}{3} + \underset{s,x}{2} + \underset{u,x}{1} + \underset{u,t}{3}$$

$$= 9$$

$$f(S,T) \le c(S,T)$$

So can determine flow is max or not if can find a cut, such that equality holds -> then flow is max

---

Try to find min cut = max flow is a theorm for this

So how do we find these?

## Residual Network

New graph $G_f$

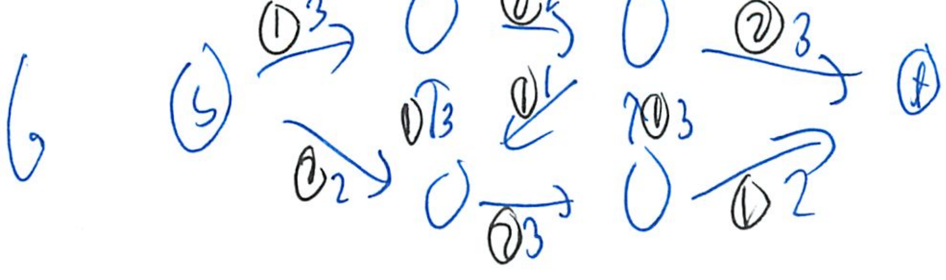
Let $f$ be flow in $G(V,E)$

So $G_f(V,E)$ is a graph w/ strictly positive residual capacites.

$$c_f(u,v) = c(u,v) - f(u,v) \geq 0$$

ie $c_f(s,u) = 3 - 2$
$= 1$

Back to original

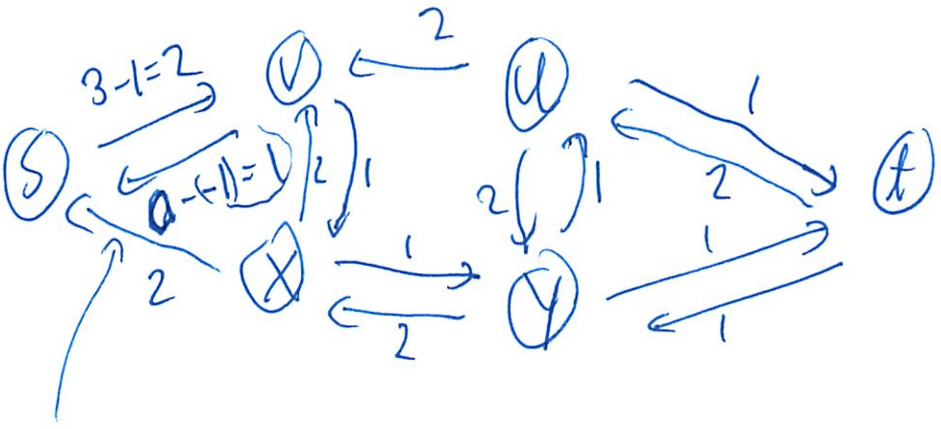


13

Can make residual flow diagram

- ~~edges~~ nodes are the same
- edges

$G_f$



Do we have edge V → S ?

$f(v,s) = -1$

$c(v,s) = 0$ ← only 1 dir arrow

Arrows are 1-way gates

$c_f(v,s) = 0-(-1)$

$= 1$

Since we can always reduce it

## Augmenting path

Any path from s to t in $G_f$

is an aug. path w.r.t. G, f

Only strictly pos capacity

Turns into reachability condition

If can find path s → t its an augmenting path and you can ↑ max flow

When ↑ value of flow, must recompute residual value of the network

Flow values can be increased along an aug path p by

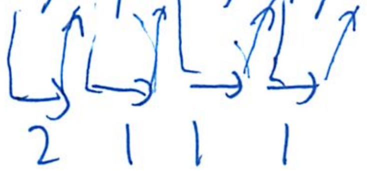
$$c_f(p) = \min_{(u,v) \in p} c_f(u,v)$$

↑ based on limiting factor

15

So our path is

s, v, x, y, t



$C_f(p) = 1$

Back to G

So we know we can ↑↓ by 1

For edges s, v

v, x

x, y

~~x~~ y, t

; so figure it out

---

What if we try $G_f'$

then we see no path s -> t

So max flow!

No proof
↳ won't be on quiz

Algorithm ↓

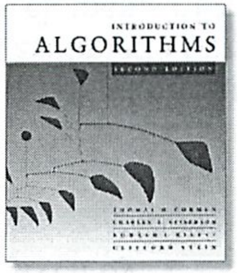
Ford-Fulkerson Alg

```
f[u,v] ← 0 for all u,v ∈ V
while an aug path p in G
↳ w.r.t f exists
    → do augment f by $c_f(p)$
```

So can build max flow alg from this

# *Design and Analysis of Algorithms*

**6.046J/18.401J**


INTRODUCTION TO ALGORITHMS

SECOND EDITION

**LECTURE 9**

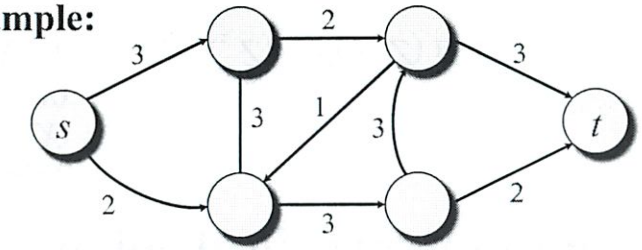
**Network Flow**

- Flow networks
- Maximum-flow problem
- Cuts
- Residual networks
- Augmenting paths
- Max-flow min-cut theorem
- Ford Fulkerson algorithm

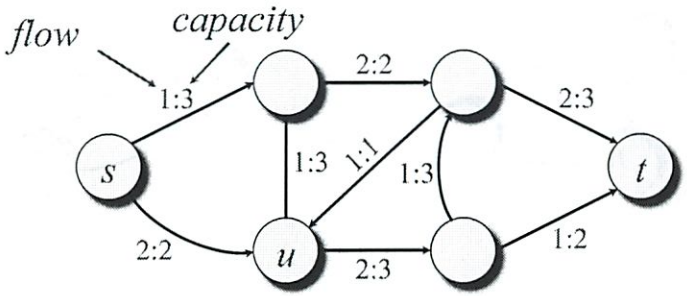
## Flow networks

**Definition.** A ***flow network*** is a directed graph $G = (V, E)$ with two distinguished vertices: a ***source*** $s$ and a ***sink*** $t$. Each edge $(u, v) \in E$ has a nonnegative ***capacity*** $c(u, v)$. If $(u, v) \notin E$, then $c(u, v) = 0$.

**Example:**



## A flow on a network



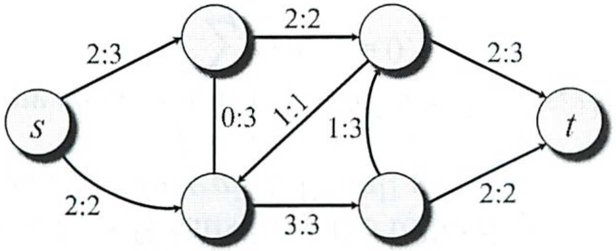
*Flow conservation* (like Kirchoff' s current law):

- Flow into $u$ is $2 + 1 = 3$.
- Flow out of $u$ is $1 + 2 = 3$.

**INTUITION:** View flow as a *rate*, not a *quantity*.

## The maximum-flow problem

**Maximum-flow problem:** Given a flow network $G$, find a flow of maximum value on $G$.



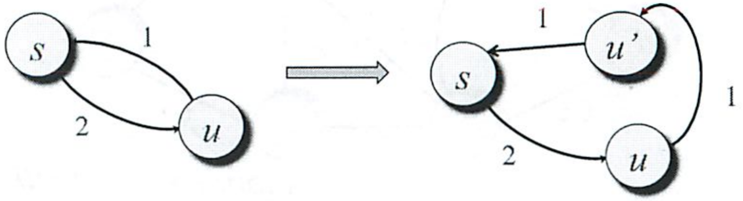
The value of the maximum flow is 4.

10/4

# Flow network Assumptions

**Assumption.** If edge $(u, v) \in E$ exists, then $(v, u) \notin E$.

**Assumption.** No self-loop edges $(u, u)$ exist



# Net Flow

**Definition.** A ***(net) flow*** on $G$ is a function $f : V \times V \to \mathrm{R}$ satisfying the following:

- ***Capacity constraint:*** For all $u, v \in V$,
$$f(u, v) \le c(u, v).$$
- ***Flow conservation:*** For all $u \in V - \{s, t\}$,
$$\sum_{v \in V} f(u, v) = 0.$$
- ***Skew symmetry:*** For all $u, v \in V$,
$$f(u, v) = -f(v, u).$$

**Note:** CLRS defines positive flows and net flows; these are equivalent for our flow networks obeying our assumptions.

# Notation

**Definition.** The ***value*** of a flow $f$, denoted by $|f|$, is given by

$$\begin{aligned} |f| &= \sum_{v \in V} f(s, v) \\ &= f(s, V). \end{aligned}$$

**Implicit summation notation:** A set used in an arithmetic formula represents a sum over the elements of the set.

- **Example** — flow conservation: $f(u, V) = 0$ for all $u \in V - \{s, t\}$.

# Simple properties of flow

**Lemma.**

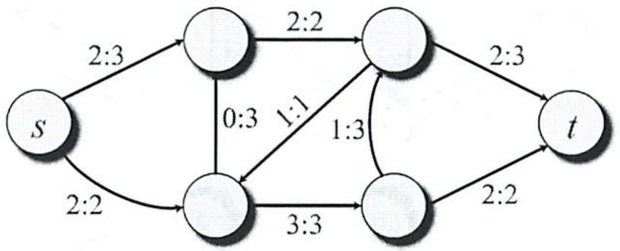
- $f(X, X) = 0$,
- $f(X, Y) = -f(Y, X)$,
- $f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$ if $X \cap Y = \emptyset$. □

**Theorem.** $|f| = f(V, t)$.

*Proof.*

$$\begin{aligned} |f| &= f(s, V) \\ &= f(V, V) - f(V{-}s, V) \qquad \textit{Omit braces.} \\ &= f(V, V{-}s) \\ &= f(V, t) + f(V, V{-}s{-}t) \\ &= f(V, t). \quad \square \end{aligned}$$

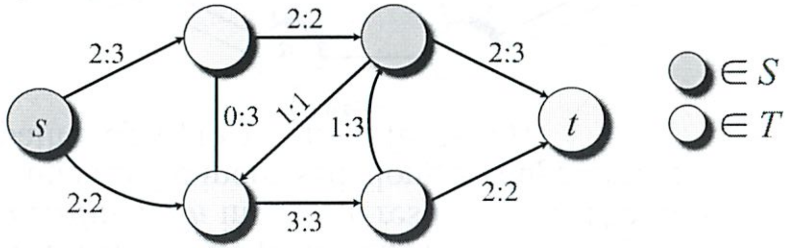
# Flow into the sink



$|f| = f(s, V) = 4$ $\qquad$ $f(V, t) = 4$

# Cuts

**Definition.** A ***cut*** $(S, T)$ of a flow network $G = (V, E)$ is a partition of $V$ such that $s \in S$ and $t \in T$. If $f$ is a flow on $G$, then the ***flow across the cut*** is $f(S, T)$.



$$f(S, T) = (2 + 2) + (-2 + 1 - 1 + 2) = 4$$

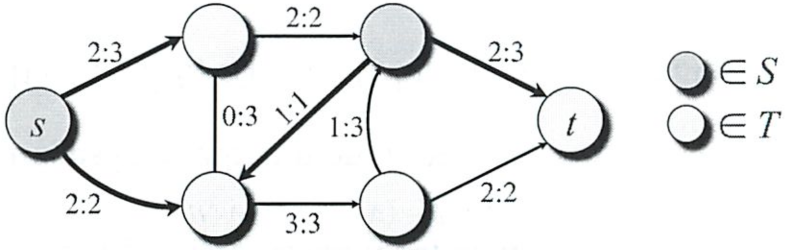
# Another characterization of flow value

**Lemma.** For any flow $f$ and any cut $(S, T)$, we have $|f| = f(S, T)$.

*Proof.*
$$\begin{aligned} f(S, T) &= f(S, V) - f(S, S) \\ &= f(S, V) \\ &= f(s, V) + f(S-s, V) \\ &= f(s, V) \\ &= |f|. \quad \square \end{aligned}$$

# Capacity of a cut

**Definition.** The ***capacity of a cut*** $(S, T)$ is $c(S, T)$.



$$c(S, T) = (3 + 2) + (1 + 3) = 9$$

# Upper bound on the maximum flow value

**Theorem.** The value of any flow is bounded above by the capacity of any cut.

*Proof.*

$$
\begin{aligned}
|f| &= f(S,T) \\
&= \sum_{u\in S}\sum_{v\in T} f(u,v) \\
&\le \sum_{u\in S}\sum_{v\in T} c(u,v) \\
&= c(S,T)\,. \qquad \square
\end{aligned}
$$

# Residual network

**Definition.** Let $f$ be a flow on $G = (V, E)$. The ***residual network*** $G_f(V, E_f)$ is the graph with strictly positive ***residual capacities***

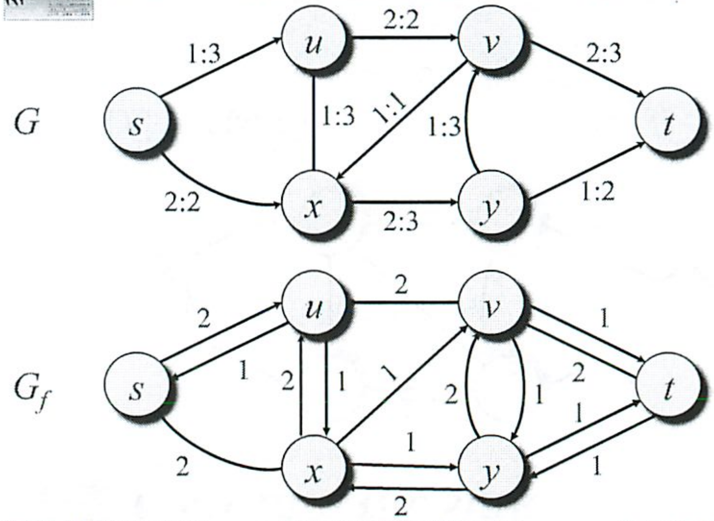
$$c_f(u, v) = c(u, v) - f(u, v) > 0.$$

Edges in $E_f$ admit more flow.

If $(v, u) \notin E$, $c(v, u) = 0$, but $f(v, u) = -f(u, v)$.

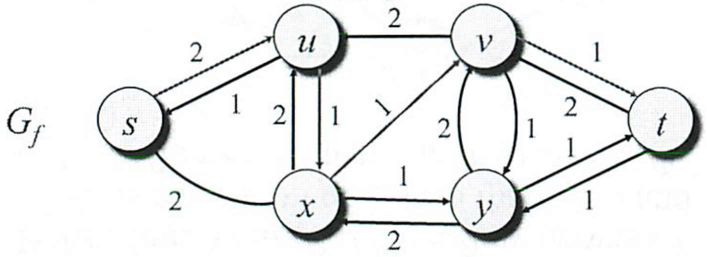
$|E_f| \le 2\,|E|$.

# Flow and Residual Network



# Augmenting paths

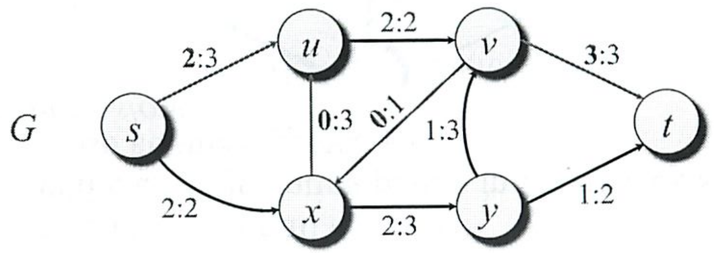
**Definition.** Any path from $s$ to $t$ in $G_f$ is an ***augmenting path*** in $G$ with respect to $f$. The flow value can be increased along an augmenting path $p$ by $c_f(p) = \min_{(u,v)\in p} \{c_f(u,v)\}$.



$p = \{s, u, x, v, t\}$, $c_f(p) = 1$

# Augmented Flow Network

$p = \{s, u, x, v, t\},\ c_f(p) = 1$



The value of the maximum flow is 4.

Note: Some flows on edges *decreased*.

# Max-flow, min-cut theorem

**Theorem.** The following are equivalent:

*1.* $|f| = c(S, T)$ for some cut $(S, T)$.
*2.* $f$ is a maximum flow.
*3.* $f$ admits no augmenting paths.

*Proof.* Next time!

# Ford-Fulkerson max-flow algorithm

**Algorithm:**

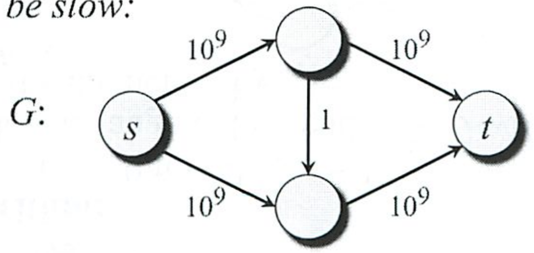
```
$f[u, v] \leftarrow 0$ for all $u, v \in V$
while an augmenting path $p$ in $G$ wrt $f$ exists
    do augment $f$ by $c_f(p)$
```

# Ford-Fulkerson max-flow algorithm

**Algorithm:**

```
$f[u, v] \leftarrow 0$ for all $u, v \in V$
while an augmenting path $p$ in $G$ wrt $f$ exists
    do augment $f$ by $c_f(p)$
```

*Can be slow:*

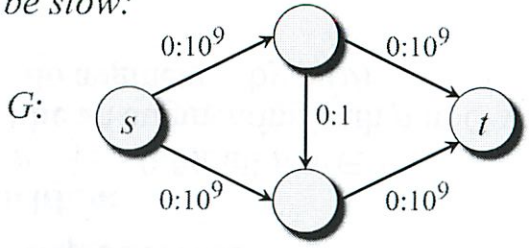


# Ford-Fulkerson max-flow algorithm

**Algorithm:**

```
$f[u, v] \leftarrow 0$ for all $u, v \in V$
while an augmenting path $p$ in $G$ wrt $f$ exists
    do augment $f$ by $c_f(p)$
```

*Can be slow:*

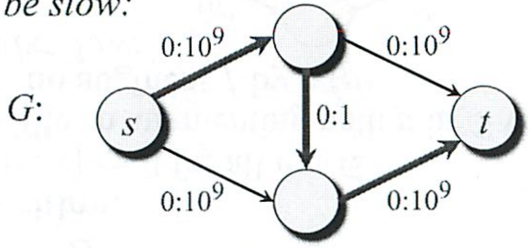


# Ford-Fulkerson max-flow algorithm

**Algorithm:**

```
$f[u, v] \leftarrow 0$ for all $u, v \in V$
while an augmenting path $p$ in $G$ wrt $f$ exists
    do augment $f$ by $c_f(p)$
```

*Can be slow:*

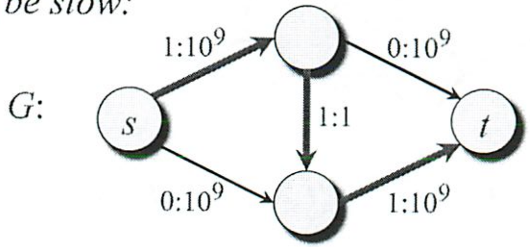


# Ford-Fulkerson max-flow algorithm

**Algorithm:**

```
$f[u, v] \leftarrow 0$ for all $u, v \in V$
while an augmenting path $p$ in $G$ wrt $f$ exists
    do augment $f$ by $c_f(p)$
```

*Can be slow:*

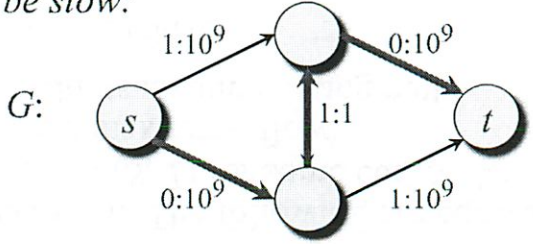


# Ford-Fulkerson max-flow algorithm

**Algorithm:**

```
$f[u, v] \leftarrow 0$ for all $u, v \in V$
while an augmenting path $p$ in $G$ wrt $f$ exists
    do augment $f$ by $c_f(p)$
```

*Can be slow:*

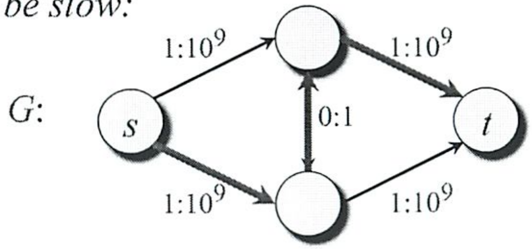


# Ford-Fulkerson max-flow algorithm

**Algorithm:**

```
$f[u, v] \leftarrow 0$ for all $u, v \in V$
while an augmenting path $p$ in $G$ wrt $f$ exists
    do augment $f$ by $c_f(p)$
```

*Can be slow:*

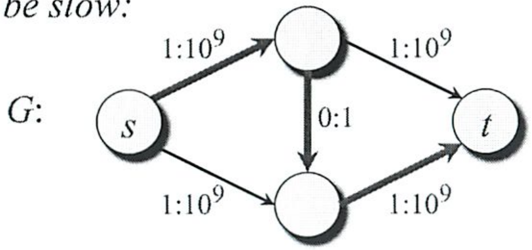


# Ford-Fulkerson max-flow algorithm

**Algorithm:**

```
$f[u, v] \leftarrow 0$ for all $u, v \in V$
while an augmenting path $p$ in $G$ wrt $f$ exists
    do augment $f$ by $c_f(p)$
```

*Can be slow:*

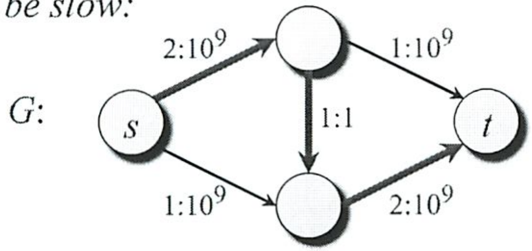


# Ford-Fulkerson max-flow algorithm

**Algorithm:**

```
$f[u, v] \leftarrow 0$ for all $u, v \in V$
while an augmenting path $p$ in $G$ wrt $f$ exists
    do augment $f$ by $c_f(p)$
```

*Can be slow:*

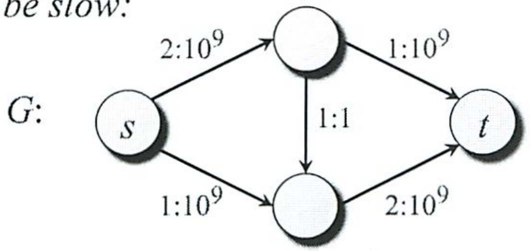


# Ford-Fulkerson max-flow algorithm

**Algorithm:**

```
$f[u, v] \leftarrow 0$ for all $u, v \in V$
while an augmenting path $p$ in $G$ wrt $f$ exists
    do augment $f$ by $c_f(p)$
```

*Can be slow:*



2 billion iterations on a graph with 4 vertices!

# 6.046

10/5

## Recitation

## Announcements

Quiz 1 on Thur 10/11

P-Set 2 due tonight

Quiz review OH Wed 7-9PM

Reg OH Tue 7-9 PM

Quiz covers all topics in reading that was listed as well

└ ie kruskal's alg

min weight single pair

Today - MST (Kruskal's Alg)
- Network flow

---

MST problem

Given weighted graph G, find a tree T that covers all vatice and has minimal weight

$$w(T) = \sum_{e \in T} w(e)$$

So greedy works

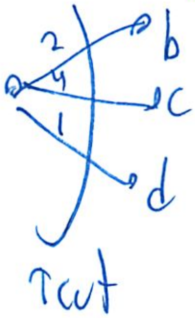
- Set of edges A
- keep adding until get tree

Adding 1 edge at a time

How do you choose which to add?

③

## Prim's alg



↙ pick the one w/ min weight

(review in more detail)

So add d

Cut $(A, V \setminus A)$

↑Chose min ~~any~~ edge

## kruskal's alg

diff approach
also greedy

Choose edges in ↑ order

1. Have a forest and keep connecting until got tree

④

7 trees

Sort in ↑ order

Select smallest edge that connects 2 forests

if a node already connected, skip it,
so no cycles

Greedy
but still special way of choosing next node/edge

Initialize empty set A
for each vertex $v \in V$
Make-set (v)

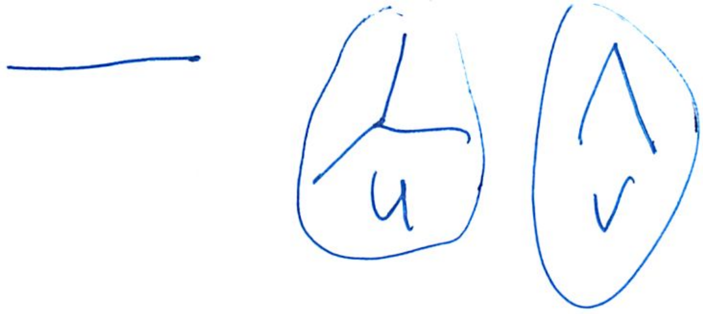
(5)

Sort all edges into non-decreasing order
- increasing or same weight

For each edge in sorted order
(u, v)

First check if already belong to same tree

```
if Find-set(u) ≠ Find-set(v):
    join (u,v)
    add (v,v) to A
```



join trees together
how → depends on substructure

How connect depends on data structure
└ Won't go into details today

$$T = T_{sort}(E) + O(E) \cdot T_{find\ set} + O(V) \cdot (T_{make\ set} + T_{union})$$

## Network Flow

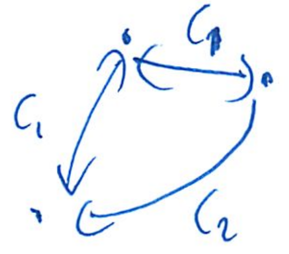
Graph G - directed

- each edge has capacity $C_{u,v} \geq 0$
- if edge $(u,v) \notin E$
  $C_{u,v} = 0$

Can have non neg capacity only in 1 direction

①

But we can ~~[illegible]~~ add a third vertex



A flow f on G is a fn

$$V \times V \to R$$

That satisfies ~~capacity~~

Capacity $f(u,v) \le c(u,v)$ for all $(u,v) \in V \times V$

Flow conservation $\sum_{v \in V} f(u,v) = 0$ $\forall u \in V / \{s,t\}$

Can Combine to super source + super sink

So 2 nodes

transform to do multiple sinks / sources

Skew symmetry $f(u,v) = -f(v,u)$

$\forall u, v \in V$

$$|f| = \sum_{v \in V} f(s,v) = f(s,V)$$

---

Fulkerson - Alg

max flow, min cut theory

the following statements are equilent

↓ capacity of min flow

1. $|f| = c(S,T)$ for some cut $(S,T)$
2. $f$ is max, min flow
3. $f$ admits no augmenting path
   - for finding augmented paths
   - So we can ↑ total flow

## Ford-Fulkerson

Was in book - are responsible for

```
$f(u,v) \leftarrow 0$ for all $u, v \in V$
while augmenting path $p$, exists in
  the residual network $G_f$
    - augment the flow by $c_f(p)$
```

Running time:

(10)

Note → doesn't always work

when does it terminate?

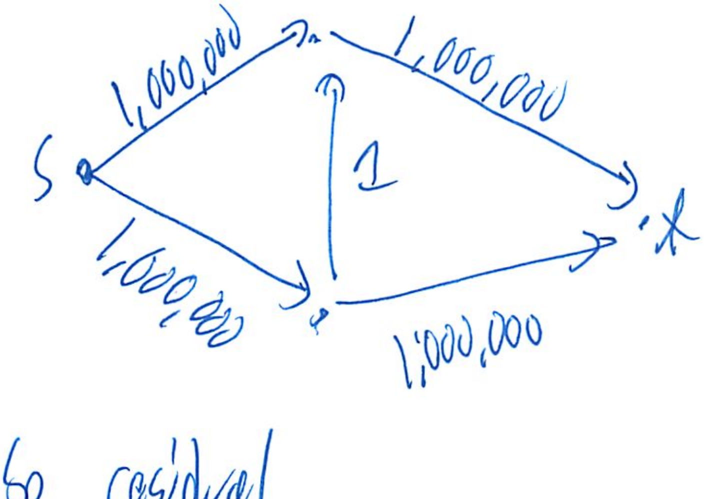
if f is unbounded → would not terminate

it is possible it won't reach max flow

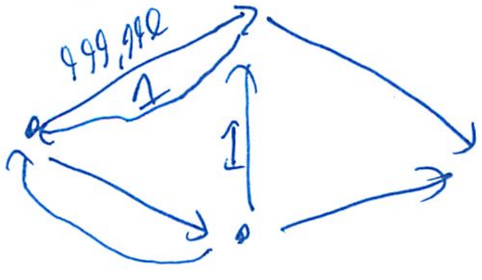
if tts are irrational

So assume integer capacites

$|f|_{max} \cdot O(E)$



So residual



Can use Edmund karp toa avoid

└ not covered

Use BFS to find augmenting paths

---

Quiz

Review Lectures, Recitation, Readings

All old quizes are online

Last 3 years are the same

Before that eaiser

10/5

*Design and Analysis of Algorithms*
Massachusetts Institute of Technology
Profs. Srini Devadas and Ronitt Rubinfeld

Oct 5, 2012
6.046J/18.410J
Recitation 4

# Recitation 4: Kruskal's Algorithm, Network Flow

## 1 Kruskal's Algorithm

### Generic MST Algorithm

Both Kruskal's Algorithm and Prim's Algorithm follows a generic MST algorithm. The generic algorithm maintains a set of edges $A$ such that $A$ is a subset of some spanning tree throughout the algorithm. While $A$ does not form a spanning tree, the algorithm finds an edge that can be added to $A$ without violating the invariant that $A$ is a subset of some spanning tree.

The main difference between Kruskal's Algorithm and Prim's Algorithm is in how they grow the set $A$. In Prim's algorithm, $A$ always form a single tree, whereas in Kruskal's algorithm, $A$ is a forest.

### Kruskal's Algorithm

Kruskal's Algorithm needs to use a data structure that supports the following operations: CREATE-SET($x$), FIND-SET($x$), and UNION($x, y$). You can read Chapter 21 of CLRS on data structures that support these two operations. Here is the pseudocode for Kruskal's Algorithm.

**Algorithm 1** Kruskal's Algorithm

```
Initialize empty set $A$
for each vertex $v \in V$ do
  MAKE-SET($v$)
end for
sort all edges into nondecreasing order by weight
for each edge $(u, v)$ taken in sorted order do
  if FIND-SET($u$) $\neq$ FIND-SET($v$) then
    $A = A \cup \{(u, v)\}$
    UNION($u, v$)
  end if
end for
return $A$
```

**Running time.** The running time depends on the data structure and is given by the following equation.

$$T_{total} = T_{sort}(E) + O(E) \cdot T_{findset} + O(V) \cdot (T_{makeset} + T_{union})$$

If we use linked-list for the data structure, the total time would be $O(T_{sort}(E) + V \log V + E) = O(T_{sort}(E) + V \log V)$

## 2 Network Flow

**Definition 1.** *A* flow networks *is a directed graph with two special nodes: a source $s$ and a sink $t$. Each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v)$. If $(u, v) \notin E$, then $c(u, v) = 0$.*

A flow on $G$ is a function $f : V \times V \to \mathbb{R}$ satisfying the following constraints:

**Capacity Constraint:** $f(u, v) \leq c(u, v)$ for all $u, v \in V$

**Flow Conservation:** $\sum_{v \in V} f(u, v) = 0$ for all $u \in V - \{s, t\}$

**Skew Symmetry:** $f(u, v) = -f(v, u)$ for all $u, v \in V$

The value of the flow is given by

$$|f| = \sum_{v \in V} f(s, v) = f(s, V)$$

In the maximum-flow problem, we want to find a flow such that $|f|$ is maximized.

**Definition 2.** *The residual network $G_f(V, E_f)$ is the graph with strictly positive residual capacities*

$$c_f(u, v) = c(u, v) - f(u, v) > 0$$

*Edges in $E_f$ are edges that can admit more flow.*

**Definition 3.** *An augmenting path in $G$ with respect to $f$ is any path from $s$ to $t$ in $G_f$. The flow can be increased along an augmenting path $p$ by $c_f(p) = \min_{(u,v) \in p}\{c_f(u, v)\}$*

**Theorem 1** (Max-Flow Min-Cut Theorem). *The following statements are equivalent:*

1. $|f| = c(S,T)$ *for some cut* $(S,T)$
2. $f$ *is a maximum flow*
3. $f$ *admits no augmenting paths*

### Ford-Fulkerson Algorithm

Max-flow algorithm can be divided into augmenting-path and push-relabel algorithm. Ford-Fulkerson is the simplest augmenting-path algorithm. Here is the pseudocode for the algorithm.

**Algorithm 2** Ford-Fulkerson Algorithm

```
$f(u,v) \leftarrow 0$ for all $u, v \in V$
while an augmenting path $p$ exists in the residual network $G_f$ do
    augment the flow by $c_f(p)$
end while
```

**Running Time.** Let $f^*$ be a maximum flow. If we only have integer capacities, at most $|f^*|$ iterations of the while loop got executed. Finding a path in the residual network using depth-first search or breadth-first search takes $O(E)$. So the total running time is $O(E|f^*|)$.

Ford-Fulkerson can be slow even with small number of vertices. Besides, the algorithm might not terminate if the edge capacities are irrational numbers.

Original 10/1/12

*Design and Analysis of Algorithms*
Massachusetts Institute of Technology
Profs. Srini Devadas and Ronitt Rubinfeld

September 25, 2012
6.046J/18.410J
Handout 7

# Problem Set 2

This problem set is due **at 11:59pm** on **Friday, October 5, 2012.**

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered by the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation section, the date and the names of any students with whom you collaborated.

Each problem must be turned in separately.

You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of the essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudo-code.
2. A proof (or indication) of the correctness of the algorithm.
3. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Full credit will be given only to correct solutions *which are described clearly*. Convoluted and obtuse descriptions will receive low marks.

---

**Exercise 2-1.** Do Exercise 5.3-2 in CLRS.

**Exercise 2-2.** Do Exercise 7.3-2 in CLRS.

**Exercise 2-3.** Do Exercise 23.1-5 in CLRS.

**Exercise 2-4.** Do Exercise 23.1-10 in CLRS.

**Exercise 2-5.** Do Exercise 23.2-4 in CLRS.

**Exercise 2-6.** Do Exercise 23.2-8 in CLRS.

**Exercise 2-7.** Do Exercise 25.1-9 in CLRS.

**Exercise 2-8.** Do Exercise 25.3-6 in CLRS.

---

**Problem 2-1. Set Elements**

Let $A$ be a set with distinct elements in the range of $[1, 100n]$. Give an efficient algorithm that runs in time $O(n \log n)$ to find whether there is some triple of three distinct elements $(x, y, z)$ such that $z = x + y$, $z = x + y + 1$, or $z = x + y - 1$.

**Problem 2-2. Briefcase Game**

Consider the following gameshow concept, inspired by "Deal or No Deal." You are given a table with $n$ identical briefcases, each with a cash prize inside of an unknown amount. The rules of the game are as follows. You are allowed to open one briefcase at a time and look at how much money is inside. At that point, you must choose whether to keep this cash prize or move on to the next briefcase. Once you reject a briefcase, you may not claim its cash anymore. Your goal is to come up with a strategy to choose the briefcase with the largest cash prize. A strategy is an algorithm which attempts to maximize some profit in the presence of an adversary, which in turn is an algorithm that attempts to minimize your profit. In this case, the game show producer is your adversary.

**(a)** Show that a deterministic strategy for this gameshow is terrible. That is, show if you use a deterministic algorithm then a gameshow producer that happens to know your algorithm can always trick you into choosing the briefcase with the least amount of money.

**(b)** Show instead that with a randomized strategy you will be able to choose the best briefcase with a probability of $\frac{1}{4}$. Why do you get a different result from part (a)? *Optional*: Can you do better?

**Problem 2-3. Square Roots Mod N**

Let $N$ be a positive integer. We will be dealing with arithmetic mod $N$ in this problem. In particular, we are interested in multiplication and taking square roots mod N. This is much like taking square roots in the real numbers, except that the roots must be integers. For example, if $N = 5$, then $\sqrt{4} = \{2, 3\}$, since $2^2 = 3^2 = 4 \bmod 5$. However, not all numbers are squares (for example, neither 2 nor 3 are squares for $N = 5$).

First, some terminology. Define the set $Z_N = \{0, 1, \ldots, N - 1\}$ that represents an arithmetic set where (1) integers may be multiplied mod N and (2) multiplication is associative. A special property is that if $\gcd(a, N) = 1$ for all $a \in Z_N$ then all elements of $a$ have inverses, i.e. there exists an element $a^{-1}$ such that $a^{-1}a = 1 \bmod N$ (sanity check: for what numbers $N$ does this occur?). Next, call $Z_N^* = \{a \in Z_N \mid \gcd(a, N) = 1\}$. In this set all elements have inverses, and it is closed under multiplication. $Z_N^*$ is called the multiplicative group of integers modulo N. Finally, remember that a composite number is a non-prime number (i.e. it can be factored).

Assume we have a black box for computing square roots modulo N; that is, a deterministic algorithm SQRT$(a, N)$ that, given positive integers $N$ and $a \in [1, 2, \ldots N]$, returns in $O(1)$ time an integer $x$ satisfying $a \equiv x^2 \bmod N$ if such an $x$ exists and 0 otherwise. In this problem we will use SQRT to give an expected polynomial time randomized algorithm for factoring integers.

We will also need the following facts:

**Fact 1:** If $N > 1$ is an odd composite integer that is not a perfect power (i.e., $N \neq a^b$ for any integers $a, b > 1$), then for any $k \in Z_N^*$, the equation $x^2 \equiv k \bmod N$ has either zero or at least four distinct solutions in $Z_N^*$.

**Fact 2:** Primality testing can be performed in deterministic polynomial time.

**(a)** We say that $d$ is a *non-trivial divisor* of $N$ if $d$ divides $N$ and $d$ is neither 1 nor $N$. For integers $x, y$, show that if $x^2 \equiv y^2 \bmod N$ but $x \not\equiv y \bmod N$ and $x \not\equiv -y \bmod N$, then $\gcd(x + y, N)$ and $\gcd(x - y, N)$ are non-trivial divisors of $N$.

**(b)** Give a Las Vegas algorithm that, given an integer $N > 1$, outputs "prime" if $N$ is prime and, if $N$ is composite, outputs a non-trivial divisor of $N$. Your algorithm should make use of SQRT and run in expected polynomial time. (Hint: Use the facts above, and beware of perfect powers.)

**(c)** Give a Las Vegas algorithm that, given an integer $N > 1$, outputs a list of all prime factors of $N$ (with multiplicities, and in any order). Your algorithm should still run in expected polynomial time.

**(d)** Factoring integers in polynomial time (expected or deterministic) is a longstanding open problem. What does this say about SQRT?

### Problem 2-4. Merchant Tycoon

You have been named minister of trade in the middle ages in the distant United Hackers Kingdom (UHK). The UHK has a set of $n$ major cities each with goods it wishes to sell and buy. Your job is to manage the trade network and trade caravans of the kingdom in order to increase revenues.

**(a)** Your first assignment is to improve the road system of the UKH. There are currently no roads, and you are given $D$ hackies (the units of currency) to build them. Each city with a trade route to the capital, Hackerville, will create a taxation revenue of $p$ hackies per month. Constructing a road between two cities $(i, j)$ has a cost $c_{i,j}$. Create an efficient algorithm that maximizes the monthly revenue from trade routes.

**(b)** With tax revenue flowing in, you have managed to connect all your cities to each other with roads. The time to travel between cities $(i, j)$ is $t_{i,j}$. Your next task is to manage the trade caravans of the kingdoms. You would ideally like to send one caravan between every two cities in your kingdom so that goods from one city will appear in every other city. Merchants are greedy, however. Each time a merchant visits a city he will sell off at most $q$ percent of his original stock in the black market, such that by the time the merchant reaches the destination he may have no more stock left to sell. Devise an efficient algorithm that will find if a merchant can be sent between two cities and arrive with some stock left, and if so it will do so in the fastest time possible. Your algorithm should run in $o(n^3 q^{-1})$.

**(c)** Terrible news! An attack by bandit raiders has made the roads unsafe to travel, which has greatly affected the revenue influx from your trade network. On the plus side, the caravans are no longer corrupt. Each road has a probability $p_{i,j}$ that the caravan will be destroyed. You would still like to send caravans between every two cities to keep trade alive. Give an efficient algorithm that gives the paths between cities where the probability that the caravans will be raided is minimized.

.

**(d)** The raids have caused major food shortages throughout the kingdom. Each city produces one staple that must be distributed to every other city. The king has decided that the capital is more important than the other cities, and prefers that caravans travel through them to keep its food stocks full. However, if the caravan has a probability of more than $m$ of getting raided while traveling through the capital then it is preferable to not send the caravan through there. If no path with overall probability less than $m$ is found, then the caravan should not be sent at all. Redesign your algorithm from part (c) to take this new constraint into account.

Revised 10/4/12

*Design and Analysis of Algorithms*
Massachusetts Institute of Technology
Profs. Srini Devadas and Ronitt Rubinfeld

September 25, 2012
6.046J/18.410J
Handout 7

# Problem Set 2

This problem set is due **at 11:59pm** on **Friday, October 5, 2012.**

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered by the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation section, the date and the names of any students with whom you collaborated.

Each problem must be turned in separately.

You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of the essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudo-code.

2. A proof (or indication) of the correctness of the algorithm.

3. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Full credit will be given only to correct solutions *which are described clearly*. Convoluted and obtuse descriptions will receive low marks.

---

**Exercise 2-1.** Do Exercise 5.3-2 in CLRS.

**Exercise 2-2.** Do Exercise 7.3-2 in CLRS.

**Exercise 2-3.** Do Exercise 23.1-5 in CLRS.

**Exercise 2-4.** Do Exercise 23.1-10 in CLRS.

**Exercise 2-5.** Do Exercise 23.2-4 in CLRS.

**Exercise 2-6.** Do Exercise 23.2-8 in CLRS.

**Exercise 2-7.** Do Exercise 25.1-9 in CLRS.

**Exercise 2-8.** Do Exercise 25.3-6 in CLRS.

---

**Problem 2-1. Set Elements**

✓ changed

Let $A$ be a finite set with $n$ distinct integer elements in the range of $[1, 10n]$. Give an efficient algorithm that runs in time $O(n \log n)$ to find whether there is some triple of three distinct elements $(x, y, z)$ such that $z = x + y$, $z = x + y + 1$, or $z = x + y - 1$.

**Problem 2-2. Briefcase Game**

Consider the following gameshow concept, inspired by "Deal or No Deal." You are given a table with $n$ identical briefcases, each with a cash prize inside of an unknown amount. The rules of the game are as follows. You are allowed to open one briefcase at a time and look at how much money is inside. At that point, you must choose whether to keep this cash prize or move on to the next briefcase. Once you reject a briefcase, you may not claim its cash anymore. Your goal is to come up with a strategy to choose the briefcase with the largest cash prize. A strategy is an algorithm which attempts to maximize some profit in the presence of an adversary, which in turn is an algorithm that attempts to minimize your profit. In this case, the game show producer is your adversary.

**(a)** Show that a deterministic strategy for this gameshow is terrible. That is, show if you use a deterministic algorithm then a gameshow producer that happens to know your algorithm can always trick you into choosing the briefcase with the least amount of money.

**(b)** Show instead that with a randomized strategy you will be able to choose the best briefcase with a probability of $\frac{1}{4}$. Why do you get a different result from part (a)? *Optional*: Can you do better?

**Problem 2-3. Square Roots Mod N**

Let $N$ be a positive integer. We will be dealing with arithmetic mod $N$ in this problem. In particular, we are interested in multiplication and taking square roots mod N. This is much like taking square roots in the real numbers, except that the roots must be integers. For example, if $N = 5$, then $\sqrt{4} = \{2, 3\}$, since $2^2 = 3^2 = 4 \bmod 5$. However, not all numbers are squares (for example, neither 2 nor 3 are squares for $N = 5$).

First, some terminology. Define the set $Z_N = \{0, 1, \ldots, N-1\}$ that represents an arithmetic set where (1) integers may be multiplied mod N and (2) multiplication is associative. A special property is that if $\gcd(a, N) = 1$ for all $a \in Z_N$ then all elements of $a$ have inverses, i.e. there exists an element $a^{-1}$ such that $a^{-1}a = 1 \bmod N$ (sanity check: for what numbers $N$ does this occur?). Next, call $Z_N^* = \{a \in Z_N \mid \gcd(a, N) = 1\}$. In this set all elements have inverses, and it is closed under multiplication. $Z_N^*$ is called the multiplicative group of integers modulo N. Finally, remember that a composite number is a non-prime number (i.e. it can be factored).

Assume we have a black box for computing square roots modulo N; that is, a deterministic algorithm $\textsc{Sqrt}(a, N)$ that, given positive integers $N$ and $a \in [1, 2, \ldots N]$, returns in $O(1)$ time an integer $x$ satisfying $a \equiv x^2 \bmod N$ if such an $x$ exists and 0 otherwise. In this problem we will use $\textsc{Sqrt}$ to give an expected polynomial time randomized algorithm for factoring integers.

We will also need the following facts:

**Fact 1:** If $N > 1$ is an odd composite integer that is not a perfect power (i.e., $N \neq a^b$ for any integers $a, b > 1$), then for any $k \in Z_N^*$, the equation $x^2 \equiv k \bmod N$ has either zero or at least four distinct solutions in $Z_N^*$.

**Fact 2:** Primality testing can be performed in deterministic polynomial time.

**(a)** We say that $d$ is a *non-trivial divisor* of $N$ if $d$ divides $N$ and $d$ is neither 1 nor $N$. Suppose $N$ is a composite integer. For integers $x, y$, show that if $x^2 \equiv y^2 \bmod N$ but $x \not\equiv y \bmod N$ and $x \not\equiv -y \bmod N$, then $\gcd(x + y, N)$ and $\gcd(x - y, N)$ are non-trivial divisors of $N$.

**(b)** Give a Las Vegas algorithm that, given an integer $N > 1$, outputs "prime" if $N$ is prime and, if $N$ is composite, outputs a non-trivial divisor of $N$. Your algorithm should make use of SQRT and run in expected polynomial time. (Hint: Use the facts above, and beware of perfect powers.)

**(c)** Give a Las Vegas algorithm that, given an integer $N > 1$, outputs a list of all prime factors of $N$ (with multiplicities, and in any order). Your algorithm should still run in expected polynomial time.

**(d)** Factoring integers in polynomial time (expected or deterministic) is a longstanding open problem. What does this say about SQRT?

**Problem 2-4. Merchant Tycoon**

You have been named minister of trade in the middle ages in the distant United Hackers Kingdom (UHK). The UHK has a set of $n$ major cities each with goods it wishes to sell and buy. Your job is to manage the trade network and trade caravans of the kingdom in order to increase revenues.

**(a)** Your first assignment is to improve the road system of the UKH. There are currently no roads, and you are given $D$ hackies (the units of currency) to build them. Each city with a trade route to the capital, Hackerville, will create a taxation revenue of $p$ hackies per month. Constructing a road between two cities $(i, j)$ has a constant cost $c$. Create an efficient algorithm that maximizes the monthly revenue from trade routes.

**(b)** With tax revenue flowing in, you have managed to connect all your cities to each other with roads. The time to travel between cities $(i, j)$ is $t_{i,j}$. Your next task is to manage the trade caravans of the kingdoms. You would ideally like to send one caravan between every two cities in your kingdom so that goods from one city will appear in every other city. Merchants are greedy, however. Each time a merchant visits a city he will sell off at most $q$ percent of his original stock in the black market, such that by the time the merchant reaches the destination he may have no more stock left to sell. Devise an efficient algorithm that will find if a merchant can be sent between two cities and arrive with some stock left, and if so it will do so in the fastest time possible. Your algorithm should run in $o(n^3 q^{-1})$.

**(c)** Terrible news! An attack by bandit raiders has made the roads unsafe to travel, which has greatly affected the revenue influx from your trade network. On the plus side, the caravans are no longer corrupt. Each road has a probability $p_{i,j}$ that the caravan will be destroyed. You would still like to send caravans between every two cities to keep trade alive. Give an efficient algorithm that gives the paths between cities where the probability that the caravans will be raided is minimized.

**(d)** The raids have caused major food shortages throughout the kingdom. Each city produces one staple that must be distributed to every other city. The king has decided that the capital is more important than the other cities, and prefers that caravans travel through them to keep its food stocks full. However, if the caravan has a probability of more than $m$ of getting raided while traveling through the capital then it is preferable to not send the caravan through there. If no path with overall probability less than $m$ is found, then the caravan should not be sent at all. Redesign your algorithm from part (c) to take this new constraint into account.

6.042
P-set 2
Initial Read

10/2
10A

#1) A = set
distinct elements

range [1, 100n]
[weird...

O(n lg n)
triple of 3 distinct elements (x,y,z)
So that

$z = x + y$
$z = x + y + 1$
or $z = x + y - 1$

So only 1 of them?
It can really be any combo
such that 2 el add together
to make a third ± 1

? What ~~[illegible]~~ are some things we learned that could fit?

dynamic programming?

look at $n \lg n$ clue

randomized?

today: greedy (missing lecture - must rewatch)

Randomized

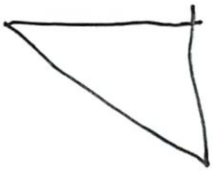
We could add each combo in $n^2$

Then compare if present $O(1)$ if hash table-like structure

But we don't need to add $a+b$ if we have $b+a$

But I think that is still $n^2$

its half the triangle



so $\frac{1}{2}n^2 \Rightarrow n^2$

13

Plus if we can go up to 100n...

But that is just $\frac{n}{100} \lg \frac{n}{100}$

Which are constant factors → distraction

Just says find if there is

but we still need to find 1 to prove it

lg n seems to involve splitting

What if we sort → n lg n

Then start in the middle and start trying stuff

Lookup O(1) or O(n)

pre layout each time

well really O(3) or O(3n)

Then we randomly back up each

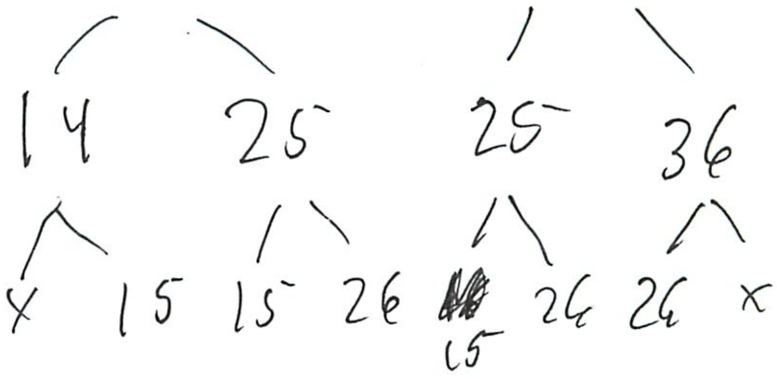
So 1 2 3 4 5 6

start 3,4

randomly move 3 left or 4 right

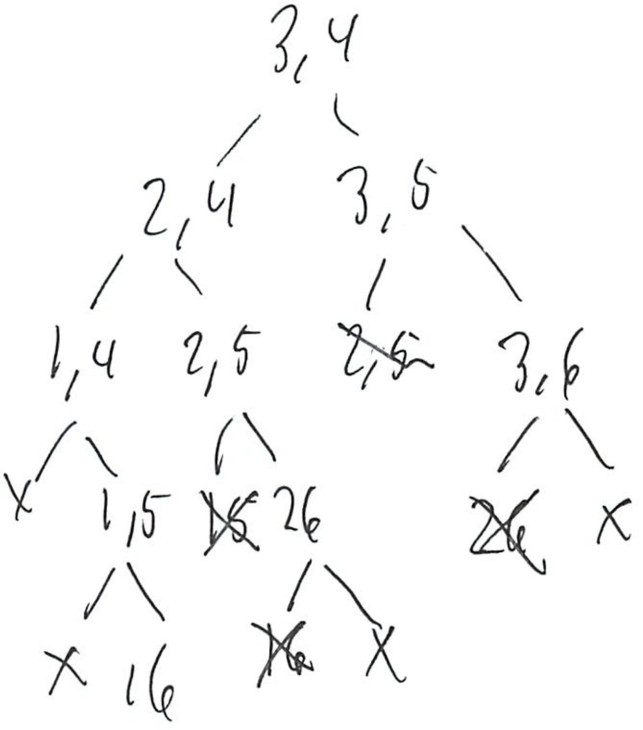
24 35

(4) Then split further



plus memoize

2T($\frac{n}{2}$) – no more complex than that



9 comparisons

n = 6

6 lg 6 = 15 something

But is this really n lg n?

$$2T(\frac{n}{2}) + O(1)$$

would be $n^{\lg_2 2}$ vs 1

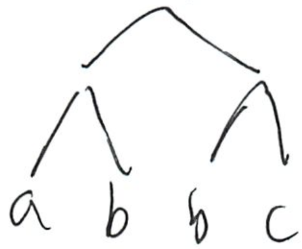
$n$ vs 1

↑

So that is not valid

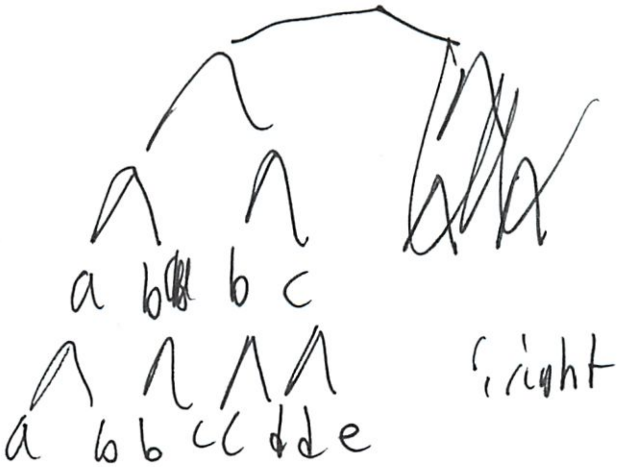
Need some notion of memoized outputs

when split



↑ repeats

also



right

Look later

10/5
9:20P

Time to copy to write up

Oh it was not really done

# of levels

Yihui very diff!

---

10/5
10:50P

Yihui's is much better

And uses FFT - which I really need to review

My ans seems very basic

---

Q It's best at representing polynomials which we have -ch

But why add them?
Some short cut

7

Oh wait binary search is exactly what I had proposed

But needed some way to add every x, y the tree

So the FFT is clearly better

add in $n \lg n$

not $n^2$

10/2
10:45A

## #2 Briefcase Game

Deal or no Deal

table w/ n breifcases

each w/ cash prize of unknown amt

Can open 1 at a time

Can keep or move on

So strategy to maximize profit

a) Show deteministic is terible

So do we know the range?

does it change from round to round?

Assuming we do

---

And deteministic means we open the Cases in a certain order

The same way everytime

And if we had a static rule such as
over 50% of running average

Then host could sort cases so that
we get as little as possible

Over hundreds of ~~cases~~ games ~~a~~ host could
figure out the pattern

And we would have no way to react

Or if he saw the alg
Where as randomized can't be as
effictivly protected againts

b) randomized strategy
<u>best</u> case w/ prob $= \frac{1}{4}$

but then we can't do knowing max
Since otherwise we would run till we
get it

③

So then if not visible what is max

We can n/e to get a sample

Then pick next one that is higher than max?

But we don't know dist

5 5 5 100 ?

Oh Deal is like

.1 .5 1 10

But where does $\frac{1}{4}$ fits in?

Since n briefcases

And then must not be consistent

from game to game

Cash prices not unique either

Wait for clarification

Also I have not downloaded a updated p-set

4

10/24
12A

So piazza @31

~~Walk~~ possible values are unknown

and pretty sure they can change from card to card

Would have to be...

The problem basically is we can't ~~st~~ know when we are ~~[illegible]~~ at max

Unless we run a avg and pick the max that is greater than that

Could we use prob?

P(is largest)

Can't reason at all about avgs

So can't connect

~~A~~ 1st P(is largest) $= \frac{1}{N}$

2nd P is largest $= 0$ or $\frac{1}{N-1}$

How does that help us at all?

① 37e ~~prizes do not need to be distinct~~

prizes are distinct

But I still have no clue how 1/4 fits in...

---

Picking does not matter → it's random

just prevents adversary from stacking the deck

So instead it's the choosing that is key

↙ keep ↓ next

And we don't know min/max/range/avg

So then: after 4 cards if new case

is max → pick it?

That gets the 4

But how is this a 1/4 shot at ~~next 2nd~~ max?

Say we have 5 cases

Pick 1 at random

~~1/2 shot below~~

most freq = 3 — no!

So can be 0, 1, 2, 3, 4 cases to left

avg = 2

same for right

2nd case has 50% shot of being greater?

No

$\frac{1}{5}\cdot 0+\frac{1}{5}\cdot 1+\frac{1}{5}\cdot 2+\frac{1}{5}\cdot 3+\frac{1}{5}\cdot 4$

$= 3$

No

Actually 50-50 is right I think

Then next time - we have 2 cases

X X

Since random 50-50 shot - ind above below each

? 50 $\frac{1}{4} \times \frac{1}{2} \times \frac{1}{4}$ ?

don't think can make that claim

esp in middle

but perhaps $\frac{1}{4}$ shot bigger than both

$\frac{1}{2} \cdot \frac{1}{2}$

Then $\frac{1}{8}$ shot bigger than 3

$\frac{1}{2^n}$

Then set = to

$\frac{1}{2^n} = \frac{1}{4}$

Solve for n

No that is prob next is larger

Not that next will be largest

$\frac{1}{N}$ shot the one you pick is largest
at start

After 1 round

$\frac{1}{N-1}$ next picked will be largest

Well we can see it

either 0 (50% ~~shot~~ time) shot largest

$\frac{1}{N-1}$ " " largest

---

So we have $\frac{1}{N-1}$ shot of picking largest next

No $\frac{1}{N}$

Then when look at

50% shot not the largest

50% shot it <u>can</u> be largest

∴ 50% $\cdot \frac{1}{N}$

9)

Same prob problems as interview

Not clear what trading!

Pre/post prob

$$P(\text{is largest} \mid < \text{prev}) = 0$$

$$p(\text{is largest} \mid > \text{prev}) = \frac{1}{n-1}$$

know large than other

So 1 2 3 4 know 1 have 2

So $\frac{1}{n - \text{less than prev}}$

$$P(A \mid B) = \frac{P(A \overset{\text{AND}}{\cap} B)}{P(B)}$$

$$P(A \cap B) = p(\text{is largest and} > \text{prev})$$

Thats what I was struggling on

They are not really ind

50% $\frac{1}{n}$

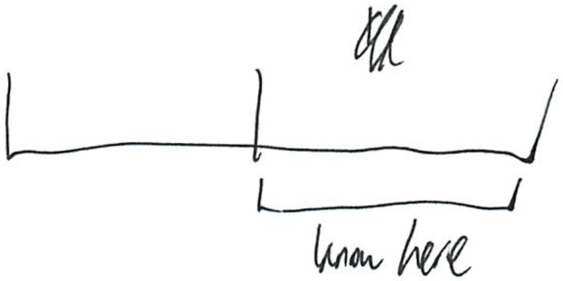
Then for n# $n=5$

$\frac{1}{n} = \frac{1}{5}$ $\frac{1}{2n} = \frac{1}{10}$

it should be less

i $\frac{1}{n/2} = \frac{2}{n}$

Since



So $1/n/2$ shot that this is largest

that sounds good

So $p(\text{is largest} \mid > \text{prev}) = \frac{1}{n/2}$

Then for $p(\text{is largest} \mid \underset{\approx \frac{1}{4}}{>} \text{prev } 2) = \frac{1}{n/4}$

(11)

Or $\left(\frac{1}{n}\right)/2 = \frac{1}{2n}$

The p is largest should ↑ as rounds go on

$$\frac{1}{n/4} = \frac{4}{n}$$

So if n=4 then a 1 shot got it
12 then 1/3 shot got it

So now

$$\frac{2^n}{n} = \frac{1}{4}$$

n is given

n = total
r = # unpached

$$\frac{2^r}{n} \leq 1/4$$

Solve for r, given n

12

$$c = \frac{\log\left(\frac{n}{4}\right)}{\log(2)}$$

So $n = 20$

$c = 2.32$

after that many times when it is the max

So lets say $n = 12$

$c = 1.58$

So 1st cand pick 1 → $\frac{1}{n}$ shot its right $\frac{1}{12}$

2nd cand " say its > prev

So $\frac{2}{n} = \frac{1}{6}$ chance its max

3rd cand > prev 2

$\frac{4}{n} = \frac{1}{3}$ chance its max

better than $\frac{1}{4}$ so take it

(13)

10/5
9:45P

So that is for *existing* cases, what about future cases?

So w/ $\frac{1}{4}$ chance it is largest
but we know that

Yuhi's is bette
but I don't get hers

Or there is prob that *next* briefcase is better

---

I'm pretty sure what I have is wrong
but I kinda like it since I came up w/ it

---

key is when we get that [illegible] max
breifcase → is it likly to be largest?
Or is one likly to be larger

I don't like my ans or Yuhi's

(14)

10/5
10:30p

r=1

For the next briefcase → $\frac{1}{4}$ shot it is bigger

not $\frac{n}{4}$ - This is ~~[illegible]~~ prob - not ~~p~~a amt

∴ Only when max

Say 10 opened of 20

$\frac{1}{10}$ chance next will be largest

but when know its lowest, prob of another largest

But we don't know range!

(I should review prob for this + interviews)

(15)

10/5
1/30

## Part

2 half

Open 1 half

P(2nd highest guy briefcase)
in look out

look at 2nd set
Only chose if higher

| | | Top | |
|---|---|---|---|
| | | A open | B |
| 2nd top | A open | x | ✓ |
| | B | won't see | might pick B |

depends which lot

Top is larger than 2nd top

This might not be best → but is $\frac{1}{4}$

is this an optimal strategy?? to max largest chance

16

Oh yes it does ask if you can do better
I mis read qu → tried to find best strat.
not just explain $\frac{1}{4}$ strat

# Integer factorization

From Wikipedia, the free encyclopedia

In number theory, **integer factorization** or **prime factorization** is the decomposition of a composite number into smaller non-trivial divisors, which when multiplied together equal the original integer.

**List of unsolved problems in computer science**

*Can integer factorization be done in polynomial time?*

When the numbers are very large, no efficient, non-quantum integer factorization algorithm is known; an effort concluded in 2009 by several researchers factored a 232-digit number (RSA-768), utilizing hundreds of machines over a span of 2 years.[1] The presumed difficulty of this problem is at the heart of widely used algorithms in cryptography such as RSA. Many areas of mathematics and computer science have been brought to bear on the problem, including elliptic curves, algebraic number theory, and quantum computing.

Not all numbers of a given length are equally hard to factor. The hardest instances of these problems (for currently known techniques) are semiprimes, the product of two prime numbers. When they are both large, for instance more than 2000 bits long, randomly chosen, and about the same size (but not too close, e.g. to avoid efficient factorization by Fermat's factorization method), even the fastest prime factorization algorithms on the fastest computers can take enough time to make the search impractical; that is, as the number of digits of the primes being factored increases, the number of operations required to perform the factorization on any computer increases drastically.

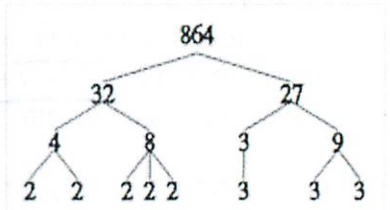
Many cryptographic protocols are based on the difficulty of factoring large composite integers or a related problem, the RSA problem. An algorithm that efficiently factors an arbitrary integer would render RSA-based public-key cryptography insecure.

## Contents

## Prime decomposition

made up of only primes?

By the fundamental theorem of arithmetic, every positive integer has a unique prime factorization. (A special case for 1 is not needed using an appropriate notion of the empty product.) However, the fundamental theorem of arithmetic gives no insight into how to obtain an integer's prime factorization; it only guarantees its existence.



This image demonstrates the prime decomposition of 864. A short-hand way of writing the resulting prime factors is $2^5 \times 3^3$

Given a general algorithm for integer factorization, one can factor any integer down to its constituent prime factors by repeated application of this algorithm. However, this is not the case with a special-purpose factorization algorithm, since it may not apply to the smaller factors that occur during decomposition, or may execute very slowly on these values. For example, if N is the number $(2^{521} - 1) \times (2^{607} - 1)$, then trial division will quickly factor 10N as 2 × 5 × N, but will not quickly factor N into its factors.

## Current state of the art

*See also: integer factorization records*

The most difficult integers to factor in practice using existing algorithms are those that are products of two large primes of similar size, and for this reason these are the integers used in cryptographic applications. The largest such semiprime yet factored was RSA-768, a 768-bit number with 232 decimal digits, on December 12, 2009.[1] This factorization was a collaboration of several research institutions, spanning two years and taking the equivalent of almost 2000 years of computing on a single-core 2.2 GHz AMD Opteron. Like all recent factorization records, this factorization was completed with a highly optimized implementation of the general number field sieve run on hundreds of machines.

### Difficulty and complexity

If a large, $b$-bit number is the product of two primes that are roughly the same size, then no algorithm has been published that can factor in polynomial time, *i.e.*, that can factor it in time $O(b^k)$ for some constant $k$. There are published algorithms that are faster than $O((1+\varepsilon)^b)$ for all positive ε, *i.e.*, sub-exponential.

The best published asymptotic running time is for the general number field sieve (GNFS) algorithm, which, for a $b$-bit number n, is:

$$O\left(\exp\left(\left(\tfrac{64}{9}b\right)^{\frac{1}{3}}(\log b)^{\frac{2}{3}}\right)\right).$$

For an ordinary computer, GNFS is the best published algorithm for large $n$ (more than about 100 digits). For a quantum computer, however, Peter Shor discovered an algorithm in 1994 that solves it in polynomial time. This will have significant implications for cryptography if a large quantum computer is ever built. Shor's algorithm takes only $O(b^3)$ time and $O(b)$ space on $b$-bit number inputs. In 2001, the first seven-qubit quantum computer became the first to run Shor's algorithm. It factored the number 15.[2]

When discussing what complexity classes the integer factorization problem falls into, it's necessary to distinguish two slightly different versions of the problem:

- The function problem version: given an integer N, find an integer d with 1 < d < N that divides N (or conclude that N is prime). This problem is trivially in FNP and it's not known whether it lies in FP or not. This is the version solved by most practical implementations.
- The decision problem version: given an integer N and an integer M with 1 ≤ M ≤ N, does N have a factor d with 1 < d < M? This version is useful because most well-studied complexity classes are defined as classes of decision problems, not function problems. This is a natural decision version of the problem, analogous to those frequently used for optimization problems, because it can be combined with binary search to solve the function problem version in a logarithmic number of queries.

It is not known exactly which complexity classes contain the decision version of the integer factorization problem. It is known to be in both NP and co-NP. This is because both YES and NO answers can be verified in polynomial time given the prime factors (we can verify their primality using the AKS primality test, and that their product is N by multiplication). The fundamental theorem of arithmetic guarantees that there is only one possible string that will be accepted (providing the factors are required to be listed in order), which shows that the problem is in both **UP** and **co-UP**.[3] It is known to be in BQP because of Shor's algorithm. It is suspected to be outside of all three of the complexity classes P, NP-complete, and co-NP-complete. It is therefore a candidate for the NP-intermediate complexity class. If it could be proved that it is in either NP-Complete or co-NP-Complete, that would imply NP = co-NP. That would be a very surprising result, and therefore integer factorization is widely suspected to be outside both of those classes. Many people have tried to find classical polynomial-time algorithms for it and failed, and therefore it is widely suspected to be outside P.

In contrast, the decision problem "is *N* a composite number?" (or equivalently: "is *N* a prime number?") appears to be much easier than the problem of actually finding the factors of *N*. Specifically, the former can be solved in polynomial time (in the number *n* of digits of *N*) with the AKS primality test. In addition, there are a number of probabilistic algorithms that can test primality very quickly in practice if one is willing to accept the vanishingly small possibility of error. The ease of primality testing is a crucial part of the RSA algorithm, as it is necessary to find large prime numbers to start with.

## Factoring algorithms

### Special-purpose

A special-purpose factoring algorithm's running time depends on the properties of the number to be factored or on one of its unknown factors: size, special form, etc. Exactly what the running time depends on varies between algorithms.

An important subclass of special-purpose factoring algorithms is the *Category 1* or *First Category* algorithms, whose running time depends on the size of smallest prime factor. Given an integer of unknown form, these methods are usually applied before general-purpose methods to remove small factors.[4] For example, trial division is a Category 1 algorithm.

This is not a good explaination

- Trial division
- Wheel factorization
- Pollard's rho algorithm

- Algebraic-group factorisation algorithms, among which are Pollard's $p - 1$ algorithm, Williams' $p + 1$ algorithm, and Lenstra elliptic curve factorization
- Fermat's factorization method
- Euler's factorization method
- Special number field sieve

### General-purpose

A general-purpose factoring algorithm, also known as a *Category 2*, *Second Category*, or *Kraitchik family* algorithm (after Maurice Kraitchik),[4] has a running time depends solely on the size of the integer to be factored. This is the type of algorithm used to factor RSA numbers. Most general-purpose factoring algorithms are based on the congruence of squares method.

- Dixon's algorithm
- Continued fraction factorization (CFRAC)
- Quadratic sieve
- General number field sieve
- Shanks' square forms factorization (SQUFOF)

### Other notable algorithms

- Shor's algorithm, for quantum computers

## Heuristic running time

In number theory, there are many integer factoring algorithms that heuristically have expected running time

$$L_n\left[1/2, 1 + o(1)\right] = e^{(1+o(1))(\log n)^{\frac{1}{2}}(\log\log n)^{\frac{1}{2}}}$$

in o and L-notation. Some examples of those algorithms are the elliptic curve method and the quadratic sieve. Another such algorithm is the **class group relations method** proposed by Schnorr,[5] Seysen,[6] and Lenstra[7] that is proved under of the Generalized Riemann Hypothesis (GRH).

## Rigorous running time

The Schnorr-Seysen-Lenstra probabilistic algorithm has been rigorously proven by Lenstra and Pomerance[8] to have expected running time $L_n\left[1/2, 1 + o(1)\right]$ by replacing the GRH assumption with the use of multipliers. The algorithm uses the class group of positive binary quadratic forms of discriminant $\Delta$ denoted by $G_\Delta$. $G_\Delta$ is the set of triples of integers $(a, b, c)$ in which those integers are relative prime.

### Schnorr-Seysen-Lenstra Algorithm

Given is an integer $n$ that will be factored, where $n$ is an odd positive integer greater than a certain constant. In this factoring algorithm the discriminant $\Delta$ is chosen as a multiple of $n$, $\Delta = -dn$, where $d$ is some positive multiplier. The algorithm expects that for one $d$ there exist enough smooth forms in $G_\Delta$. Lenstra and Pomerance show that the choice of $d$ can be restricted to a small set to guarantee the smoothness result.

Denote by $P_\Delta$ the set of all primes $q$ with Kronecker symbol $\left(\frac{\Delta}{q}\right) = 1$. By constructing a set of generators of $G_\Delta$ and prime forms $f_q$ of $G_\Delta$ with $q$ in $P_\Delta$ a sequence of relations between the set of generators and $f_q$ are produced. The size of $q$ can be bounded by $c_0(\log|\Delta|)^2$ for some constant $c_0$.

The relation that will be used is a relation between the product of powers that is equal to the neutral element of $G_\Delta$. These relations will be used to construct a so-called ambiguous form of $G_\Delta$, which is an element of $G_\Delta$ of order dividing 2. By calculating the corresponding factorization of Δ and by taking a gcd, this ambiguous form provides the complete prime factorization of $n$. This algorithm has these main steps:

Let $n$ be the number to be factored.

1. Let Δ be a negative integer with Δ = *-dn*, where *d* is a multiplier and Δ is the negative discriminant of some quadratic form.
2. Take the *t* first primes $p_1 = 2, p_2 = 3, p_3 = 5, \ldots, p_t$, for some $t \in \mathbb{N}$.
3. Let $f_q$ be a random prime form of $G_\Delta$ with $\left(\frac{\Delta}{q}\right) = 1$.
4. Find a generating set $X$ of $G_\Delta$
5. Collect a sequence of relations between set $X$ and $\{f_q : q \in P_\Delta\}$ satisfying:
$$\left(\prod_{x\in X} x^{r(x)}\right) \cdot \left(\prod_{q\in P_\Delta} f_q^{t(q)}\right) = 1$$
6. Construct an ambiguous form *(a, b, c)* that is an element $f \in G_\Delta$ of order dividing 2 to obtain a coprime factorization of the largest odd divisor of Δ in which Δ = *-4a.c* or *a(a - 4c)* or *(b - 2a).(b + 2a)*
7. If the ambiguous form provides a factorization of *n* then stop, otherwise find another ambiguous form until the factorization of *n* is found. In order to prevent useless ambiguous forms from generating, build up the 2-Sylow group $S_2(\Delta)$ *of* $G(\Delta)$.

To obtain an algorithm for factoring any positive integer, it is necessary to add a few steps to this algorithm such as trial division, Jacobi sum test.

### Expected running time

The algorithm as stated is a probabilistic algorithm as it makes random choices. Its expected running time is at most $L_n\left[1/2, 1 + o(1)\right]$.[8]

## See also

- Canonical representation of a positive integer

## Notes

1. ^ *a b* Kleinjung, et al (2010-02-18). *Factorization of a 768-bit RSA modulus* (http://eprint.iacr.org/2010/006.pdf) . International Association for Cryptologic Research. http://eprint.iacr.org/2010/006.pdf. Retrieved 2010-08-09.
2. ^ LIEVEN M. K. VANDERSYPEN, et al (2007-12-27). *NMR quantum computing: Realizing Shor's algorithm* (http://www.nature.com/nature/links/011220/011220-2.html) . Nature. http://www.nature.com/nature/links

10/2
11:10

#3) Square Roots mod n

Oh I hate these questions...

n = positive integer

arithmitic mod n

roots must be integer

ie $\sqrt{4} = \{2, 3\}$

since $2^2 = 3^2 = 4 \mod 5$

( must return all ans )

$Z_n$ = set where integers may be multiplied mod N

~~Multiplication is Associative~~

$Z_n = \{0, 1, \ldots, N-1\}$

So this is the result of the mod

ie mod 4 → $\{0, 1, 2, 3\}$

①

if $\gcd(a, N) = 1$ for all $a \in Z_n$

then all els of a have inveses

$$a^{-1} a = 1 \bmod N$$

(Qv: For what N does this occur?)

(I don't get this qv

All N?

Should test later

Next, call $Z_N^* = \{a \in Z_n \mid \gcd(a, N) = 1\}$

in this set all have inverses

Closed under multiplication

= Multiplicive g/p of integers mod N

— Composite = non prime, can be factored

— Black box to compute sq roots mod N

deteministic alg SQRT(a, N)

That given N, $a \in [1, 2, \ldots N]$

returns in $O(1)$ time an int

③

Satisfying $a \equiv x^2 \bmod N$
if $x$ exists, 0 otherwise

Use SQRT to give an expected poly time
alg for factoring ints

10/4
10:40P

Closed under multiplication = any member of set

× any other member of set (including itself)
yields another member of the set

ex) $\{0,1\}$

$0 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 0 = 0$
$1 \times 1 = 1$

✓ closed under multip

---

So how does our black box work?
don't think it matters...?
Can it ~~ax~~ actually find it?

Want to factor integers

5

More Facts

1. if $N > 1$ odd composite int that is not a perfect power $N \neq a^b$ for any int $a, b > 1$ then for any $k \in \mathbb{Z}_N^*$

   $x^2 = k \mod N$ has 0 or 4 distinct sol

   ?: why?

2. Pimality testing can be pefomed in deteministic polynomial time

Questions

a) $d$ is a non-tirval divisor of $N$

6

10/5
1:30A

I don't get this → reread!

$2^2$ mod 5 = 4

$3^2$ mod 5 = 9 → 4

$N=5$ = mod 5

Set $Z_n$

Arithmetic set = set of natural #s that can be defined by a formula of 1st order Peano arithmetic

↳ axioms for natural #s

if there is a formula $\varphi(n)$ s.t. each # n in X ~~an~~ iff $\varphi(n)$ holds

So basically we can define it someway?

What does it pratically mean?

it seems like → can be constructed

7

Assoaitive = Order does not matter

So basically $Z_n$ is all #s less then n

$gcd(a, N) = 1$

What does that mean?

gcd = greatest common denom

So like $N = 5$

$gcd(0, 5) = 5$

$1, 5 = 1$

$2, 5 = 1$

$3, 5 = 1$

$4, 5 = 1$

$5, 5 = 5$

$6, 5 = 1$

$7, 5 = 1$

$8, 5 = 1$

$9, 5 = 1$

$10, 5 = 5$

then all el of a have inverses

but 0, 5 is always 5 ?

have $a^{-1} a = 1 \mod N$

$\frac{1}{2} \cdot 2 = 1 \mod N$

any #

- well real
- and invertable

but hint says this is only ~~one~~ some N

which ?

gcd = largest pos int that divides the #s w/o a remainder

$\gcd(8, 12) = 4$

but $\gcd(a, 0) = a$

and 0 is always inc

we are not looking at ints - right?

Ok skip ahead...

$\mathbb{Z}_N^*$ = all el have inverses
closed under multiplication

Composite is non prime
can be factored

Black box

$a = x^2 \mod N$

? get $x$

Want to factor integers

Ok WP article

Ah ok So this problem in real life is unsolvable

But they have a baby version for us

(10)

4 distinct solution

Sh at least

Prime = Only divisor 1 and itself

ie 5, but not 6

$6/2=3$ $6/3=2$

a) d is non-trivial divisor of N if d divids N and d is neiter 1 nor N

$\frac{N}{d}$ $d \neq 1, N$

so integer

Suppose N = Composite integer

For int x, y Show that if $x^2 \equiv y^2 \mod N$

but $x \not\equiv y \mod N$

$x \not\equiv -y \mod N$

Then $\gcd(x+y, N)$, $\gcd(x-y, N)$ are non trivial divisors of $N$

(11)

10/5
BP
Friends

Yihui Saw

a)

: check if $y, -y$ are Friend

~~show that if that~~

show that if $x^2 = y^2$ mod $N$

$x \neq y$ $x \neq -y$

Then gcd $(x+y, N)$ are non trivial divisors (not $1, N$)

$x \sim y$

Proof by Contridiction

if $(x+y, N) = N$

Then $x+y$ is multiple of $N$

So Mod $N$ sum $= 0$

ok

So then $x = y$ mod $N$

So $(x+y, N) = 1$

rel prime

= share no comon divisors except 1

$\gcd(a,b) = 1$

Then $x^2 = y^2 \mod N$

✓ would not have seen/thought of

$x^2 - y^2 = 0$

$(x+y)(x-y) = 0$

thats how you split it up!

On closer look its factoring

but for rel prime $(x-y)=0$

~~R~~ So this is for both cases

One side must be 0

then remainder contridicts

(13)

(I am not good at this Contradicts stuff...)

(As I rewrite, I think I am getting this

Told $x+y$ must be rel prime since given

$$\boxed{\gcd(x+y, N) = 1}$$

b) Give a Las Vegas alg

randomized alg that always gives correct results

ie quicksort w/ random pivots

expected runtime = finite

Monte Carlo deterministic runtime

very small chance its wrong

Past Fall 2010 pset 2 similar

(14)

Given an int $N > 1$ ~~de~~

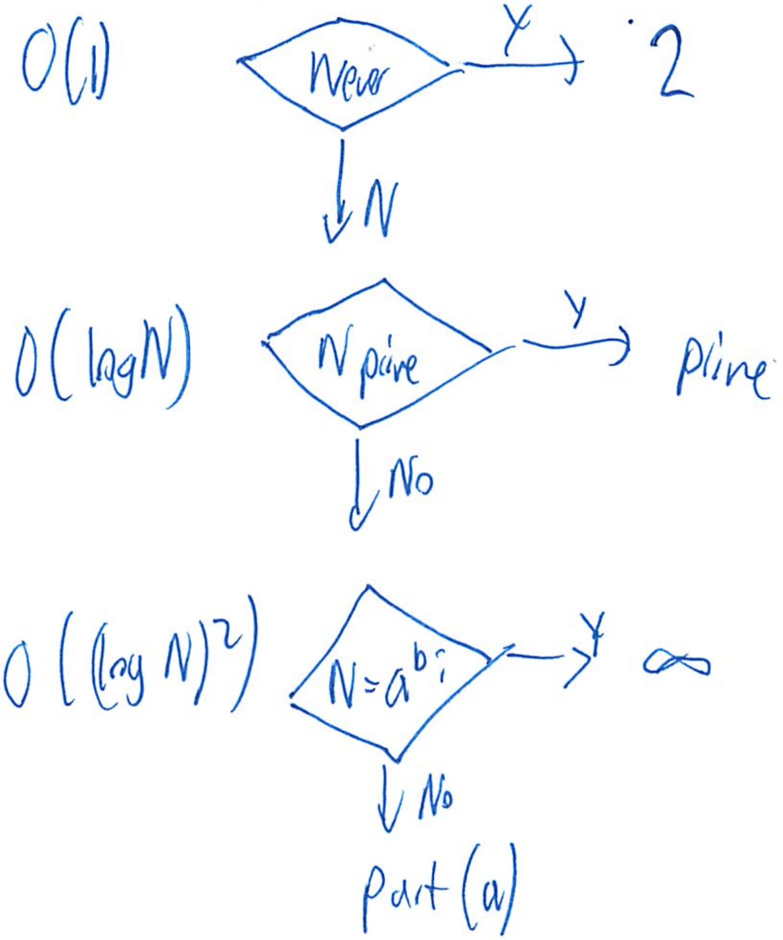
Outputs " prime "

Or if composite a non trivial divisor of N

Should use SQRT and run in poly time

---

~~Yihu~~ Yihui OH Notes



(15)

prob for A

$x^2 = y^2 \mod N$

$x \neq y$ and $x \neq -y \mod N$

↳ $\exists$ non trivial divisor of $N$

$x \in \{0, \ldots, N-1\}$ random

$Y = \text{SQRT}(x^2, N)$

↓

$x^2 = y^2 \text{ Mod } N$

$\gcd(x, N) \neq 1$ → $\gcd(x, N)$ is non trivial divisor

$= 1$

$x \in \mathbb{Z}_n^*$

10/5
8:30

Ok lets finish this

Hint: Watch for "perfect powers" - an positive integer that can be an integer of another positive intiger

ex $2^2=4$

$\underbrace{2^3}_{\text{power}}=\underbrace{8}_{\text{int}}$

of
other integer

4, 8, 9, 16, 25, 27, 32, 36, 49, 64, 81, 100, etc

---

How do you know if something is prime?

Can check 2 → $\sqrt{n}$

Primality tests
(wp article)

wp: Most do w/ probability

17

1. Pick a random # a
2. ~~Choose~~ Check some equality
3. Repeat till certain

↑ but this is Monte Carlo

Miller-Rabin
Solovay Strassen } both prob.

deterministic ≠ runs same time always
but I do' think this is a primality test

---

Fall 2010: ~~a~~ same tree as in recitation
And it loops
But I guess it tries everything
Which ~~[illegible]~~ our random algs on WP could do
random just makes them more likely to return quickly

(18)

c) LV alg

given int $N > 1$ output a list of all prime factors of $N$

i.e. so a prime factorization test

---

i. try every divisor

Or something w/ $\sqrt{}$ s

---

Fall 2010: Factor (d)

Factor $\left(\frac{N}{d}\right)$

So factor 10 = factor N

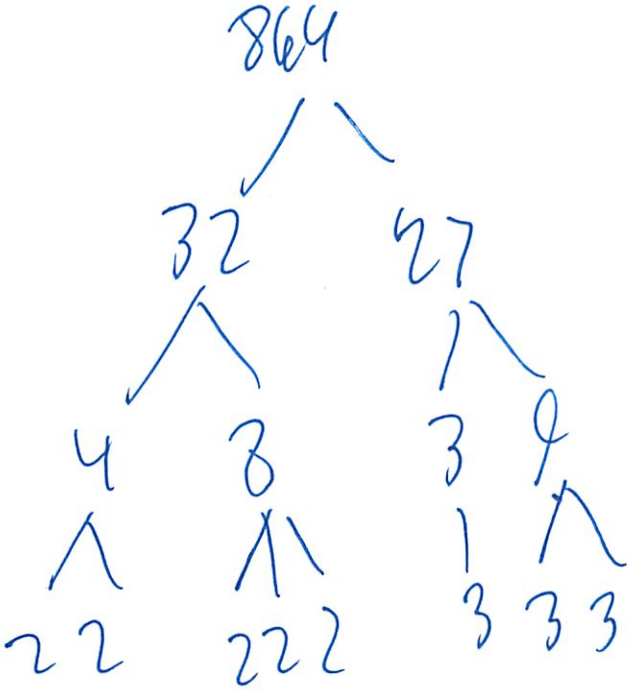
one factor = 5

So Factor (5) Factor $\frac{10}{5} = (2)$

So it basically splits

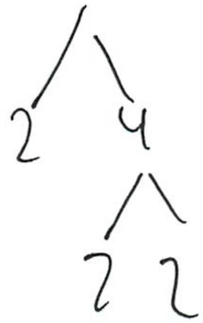
19

This is that prime factorization free I saw on WP



?? how 3

Well ours would do



Time: $\log n$ levels

$$2T(\tfrac{n}{2}) + O(n^k)$$

$n^{\log_2 2}$ Bigger

---

d) Factoring int in poly time is a long standing open problem

What does this say about SQRT

We can dream!

Fall 2010

for some prime $p > 2n$. Likewise, define

$$f_B(x) = \prod_{i=0}^{m} (x - i)^{b_i} \bmod p.$$

for multiset $B$. Note that the degree of $f_A(x)$ is just $\sum_i a_i = n$, and likewise for $f_B(x)$. Let $g(x) = f_A(x) - f_B(x)$, which has degree $\leq n$. Observe that $A$ and $B$ are identical if and only if $g(x) \equiv 0 \bmod p$ (i.e., all coefficients of $g(x)$ are $0 \bmod p$). To check for the latter, we pick a random $r$ from $\mathbb{Z}_p$ and evaluate $g(r)$. By the Schwartz-Zippel Lemma, if $g(x) \not\equiv 0 \bmod p$, then the probability that $g(r) \equiv 0 \bmod p$ is at most $n/p < 1/2$ by our choice of $p$. Thus by returning "identical" if and only if $g(r) \equiv 0 \bmod p$, we have a Monte Carlo algorithm with (one-sided) error probability $< 1/2$.

The running time is dominated by a single evaluation of $f_A(r)$ or $f_B(r)$. But observe that to evaluate $f_A(r)$, say, we can simply go through each element of $A$, and whenever we see an $i$ multiply a running product by $(r - i)$. Since we can perform arithmetic operations in constant time, this is a constant time procedure per element. As a result, evaluation takes $O(n)$ time, so our algorithm also has a running time of $O(n)$.

**Problem 2-4. Factoring**

Let $N$ be a positive integer. Recall that $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$ and $\mathbb{Z}_N^* = \{a \in \mathbb{Z}_N \mid \gcd(a, N) = 1\}$. Assume we have a black box for computing square roots modulo N; that is, a deterministic algorithm $\text{SQRT}(a, N)$ that, given positive integers $N$ and $a \in \mathbb{Z}_N$, returns in $O(1)$ time an integer $x \in \mathbb{Z}_N$ satisfying $a \equiv x^2 \bmod N$ if such an $x$ exists and 0 otherwise. In this problem we will use SQRT to give an expected polynomial time randomized algorithm for factoring integers.

We will need the following facts:

**Fact 1:** If $N > 1$ is an odd composite integer that is not a perfect power (i.e., $N \neq a^b$ for any integers $a, b > 1$), then for any $k \in \mathbb{Z}_N^*$, the equation $x^2 \equiv k \bmod N$ has either zero or at least four distinct solutions in $\mathbb{Z}_N^*$.

**Fact 2:** Primality testing can be performed in deterministic polynomial time[1].

**(a)** We say that $d$ is a *non-trivial divisor* of $N$ if $d$ divides $N$ and $d$ is neither 1 nor $N$. For integers $x, y$, show that if $x^2 \equiv y^2 \bmod N$ but $x \not\equiv y \bmod N$ and $x \not\equiv -y \bmod N$, then $\gcd(x + y, N)$ and $\gcd(x - y, N)$ are non-trivial divisors of $N$.

**Solution:** First we prove the following lemma: if $a, b$ are integers such that $a, b \not\equiv 0 \bmod N$ but $ab \equiv 0 \bmod N$, then $\gcd(a, N)$ and $\gcd(b, N)$ are non-trivial divisors of $N$.

[1] For a proof of this interesting theorem, which is beyond the scope of this class, see *PRIMES is in P* by Manindra Agrawal, Neeraj Kayal and Nitin Saxena in 2002.

Suppose, to get a contradiction, that $\gcd(a, N) = 1$. Then $a \in \mathbb{Z}_N^*$ and has a multiplicative inverse $a^{-1}$. Multiplying both sides of $ab \equiv 0 \bmod N$ by $a^{-1}$ we get $b \equiv 0 \bmod N$, contrary to given assumption. Thus, $\gcd(a, N) \neq 1$ (or $N$, since $a \not\equiv 0 \bmod N$). By symmetry the same holds for $b$.

Returning to the main problem, since $x^2 \equiv y^2 \bmod N$, we can write

$$0 \equiv x^2 - y^2 \equiv (x + y)(x - y) \bmod N.$$

But since $y \not\equiv \pm x \bmod N$, we know $x \pm y \not\equiv 0 \bmod N$. Applying our lemma completes the proof.

**(b)** Give a Las Vegas algorithm that, given an integer $N > 1$, outputs "prime" if $N$ is prime and, if $N$ is composite, outputs a non-trivial divisor of $N$. Your algorithm should make use of SQRT and run in expected polynomial time. (Hint: Use the facts above, and beware of perfect powers.)

**Solution:**

```
ONE-FACTOR($N$)
1  **if** $N$ is prime, **return** "prime"
2  **if** $N$ is even, **return** 2
3  **if** $N = a^b$ for some integers $a, b > 1$, **return** $a$
4  Pick $x$ from $\{1, 2, \ldots, N - 1\}$ uniformly at random
5  **if** $\gcd(x, N) > 1$, **return** $\gcd(x, N)$
6  Let $a := x^2 \bmod N$
7  Let $y :=$ SQRT$(a, N)$
8  **if** either $y \equiv x \bmod N$ or $y \equiv -x \bmod N$, **goto** Line 4
9  **else return** $\gcd(x + y, N)$
```

If $N$ is prime, by Line 1 will catch it (courtesy of Fact 2). Suppose instead that $N$ is composite. By Fact 1, there are at least four values SQRT$(a, N)$ could return in Line 7 (we checked for $N$ being even or a perfect power, and there has to be at least one solution, namely, $x$). Since it is deterministic, it will always return $y$. But our $x$ was random, so $x$ was just as likely to have been one of the two (or more) that were *not* $\pm y \bmod N$. In other words, there is a probability of at least $1/2$ that the algorithm will go to Line 9. In this case, part (a) tells us that we can safely return either $\gcd(x + y, N)$ or $\gcd(x - y, N)$. If this failed, we simply repeat from Line 4, with the same probability of success ($\geq 1/2$) on each independent trial.

Each step in ONE-FACTOR runs in polynomial time, including checking if $N$ is a perfect power in Line 3: we can check each value of $b$, $2 \leq b \leq \log_2 N$, by looking for a matching $a$ via binary search (since having fixed $b$, $f(a) := a^b$ is a monotone function) within the range $2 \leq a \leq N$. Moreover, each time we execute Lines 4-8 we have a probability of at least $1/2$ of not having to repeat, so the expected running time of One-Factor is at most twice the running time of a single trial, therefore polynomial as well.

**(c)** Give a Las Vegas algorithm that, given an integer $N > 1$, outputs a list of all prime factors of $N$ (with multiplicities, and in any order). Your algorithm should still run in expected polynomial time.

**Solution:**

```
Factor($N$)
Compute $d :=$ One-Factor($N$)
if $d =$ "prime", return $d$
else return Factor($d$), Factor($N/d$)
```

This procedure clearly outputs the desired factorization of $N$. To analyze its running time, consider the recursion tree it traverses. This binary tree has as many leaves as $N$ has prime factors (with multiplicities), which is at most $\log_2 N$, so it also has at most $\log_2 N$ internal nodes. Each node or leaf represents one call to One-Factor, which takes expected polynomial time, so Factor also runs in polynomial time.

**(d)** Factoring integers in polynomial time (expected or deterministic) is a longstanding open problem. What does this say about Sqrt?

**Solution:** It is at least as hard as factoring. In other words, it doesn't exist, as far as we know.

10/4
10:50p

# #4 Merchant Tycoon

minister of trade in middle hages

United Hackers kingdom (UHk)

n major citizes

Manage trade networks

? central planning!

¿ Graph networks ?

a) improve road system

Currently no roads

Get D capital

route to capital = p per month

¿ any route or direct route?

road b/w cities (i,j) costs c up front

max rev p.

2

So this is Min Spanning tree
straight up

~~Mo~~

monthly rev does not matter
unless ya had to reinvest your profits
but no

is weighted w/ cost
want to ~~maximize~~ minimize
always take locally optimal choice

but prob can't be this simple
or the challenge is in B

→ Ahh we want max revenue
at min cost

(3)

~~$~~ But 2 sep pools of $

Max revenu that minize cost

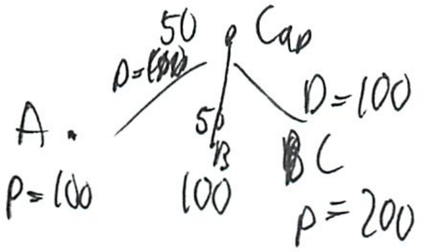
So instead do ROI?

$$\boxed{\frac{\text{Gain} - \text{Cost}}{\text{Cost}}}$$

Or just profit $\boxed{\text{gain} - \text{cost}}$

Since want max reve for 1 month

Say D=100



So ROI

50% 50% 100%

Profit 50 50 100

(why the same?

(4)

$D = 75$
$p = 100$

$$\frac{100-75}{75} = \frac{1}{3} = 33\%$$

25 profit
? do this

Stop when D hackles

Why is this best?

Well I said it - max profit

otherwise very profitable, but very expensive

? Can we prove

Profit = vertice
Cost = edge

Well just look at each edge

and trying to maximize
Still locally optimal?

(5)

b) All coads have been bilt

time $= t_{i,j}$

Want to send 1 caravan b/w every 2 cities

so $V^2$?

Merchants are greeky. Each time a merchant visits a city he will sell of at most q % so that when reaches destination may have no stock left to sell.

Alg to find if ~~So~~ merchant ~~can~~ can be sent b/w 2 cities, arrive w/ some stock left and fastest time

$o(n^3 q^{-1})$

↑ ? wtf? - inverse

50% → 2

~~lower bound~~

Upper bound

not asy tight

(6)

I don't really get

(a) 39 → all pairs shortest path
Floyd Warshall $\Theta(V^3)$

(a) 29 → need to figure out what # of edges e is given by the % q

? is this another of the prob ones?

So Floyd Warshall
w/ this percentage thing

$$O\left(\frac{n^3}{q}\right)$$

↑
upper bound
but can be less asy

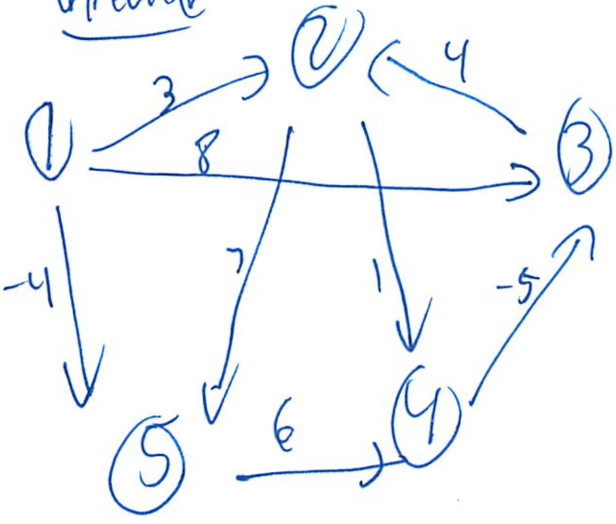
7

# Floyd Warshall

Have table of shortest paths for all pts

(Don't think I ever studied in depth, Reminds me more of 6.01 than 6.006)

directed



(Study now so know)

neg weights
no ⊖ ~~weigh~~ cycles

to ~~fro~~ ~~from~~

from ~~from~~ ~~to~~

$$d_{ij}(0) = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 3 & 8 & \infty & -4 \\ 2 & \infty & 0 & \infty & 1 & 7 \\ 3 & \infty & 4 & 0 & \infty & \infty \\ 4 & 2 & \infty & -5 & 0 & \infty \\ 5 & \infty & \infty & \infty & 6 & 0 \end{array}$$

Considers intemodiate vertex of a simple path

Can decompose into subpaths

CLRS is very Verbose

$d_{ij}^{(k)}$ is weight of shortest path $i \to j$
w/ k intemodiate nodes

$d_{ij}^{(0)} = W_{ij}$

$$d_{ij}^{(k)} = \left( \begin{cases} W_{ij} & k=0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1} \right. \end{cases} \right)$$

Operationally 3 for loops

$\Pi$ = predicessor matrix
to print shortest path

---

So that chart I drew on last pg
$= d_{ij}(0)$

Then what for next cond?

So in example

$4 \underset{\text{via } 3}{\rightarrow} 2$ is now 5

So $$d_{ij}^{(1)} = \min\left(\underset{\infty}{d_{ij}^{(0)}},\ \underset{\cancel{4}-5}{d_{43}^{(0)}} + \underset{4}{d_{32}^{(0)}}\right)$$

for $i = 4$
$j = 2$

$= -1$

But that does not happen cond 1

---

$k$ = subset of vertices

Consider intermediate where point is $k$

Still don't get what $k$ is ...

---

Look at code

is $k$ a specific intermediate node?

Ok so then for intermediate being node 1
can we get from every $i \to j$ via node 1?

(This makes so much more sense now!)

But then k in 25.5 doesn't make sense
well in set $\{1, 2, .. k\}$ of nodes
? So nodes must be numbered ?

---

Re look at code

4 → 2 via 1 is 2 + 3
So we write that
but try $n^2$ of them
many still impossible

(I don't really get how this is the best way → relaxing better? Dijkstra)
L only single source

Ah - but still so much sense

Can I go through pt 1<br>
2<br>
3<br>
etc

After gone through all nodes → done

---

Back to q√

q percent of original stock

~~Only b/w~~ 2 cities → any 2 cities

minimize $A_{ij}$ → the cost/weight of the roads

~~m~~ minimize $q$

$A_{ij}$ = ~~[illegible]~~ edge

$q$ = node

No, real trade off ?

So just Floyd Warshall

Remember @29 What # edges is given ~~#~~ % q

(this was so poorly explained!)

I don't get this clarification

? A second step after Warshall

but then how is it $\frac{n3}{q}$ ?

---

Im ignoring

---

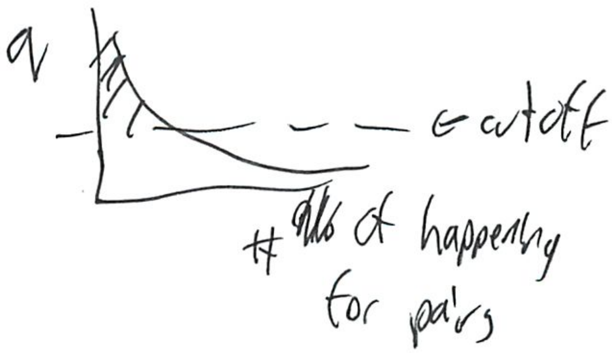
Oh some stuff left

So some additional cost

So at each level of q cuts some at

like a limiting factor

Say



(13)

c) Attack by bandits

↓ revenue

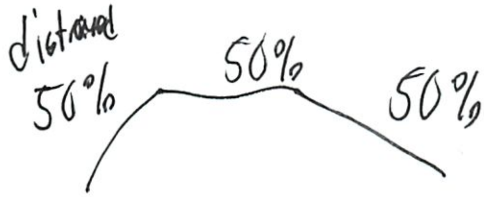
road prob $p_{i,j}$ will be destroyed

So prob caravans raided minimized

F-W again

but whats the trick?

Or do we multiply prob



$1-(.50 \cdot .50 \cdot .50) = .125\%$ chance route survives

Q29 looks similar

My ans to b was simple
just took me a while to review F-W

(14)

d) Food shortage

every city has a product for other city

So $n^2$ caravans

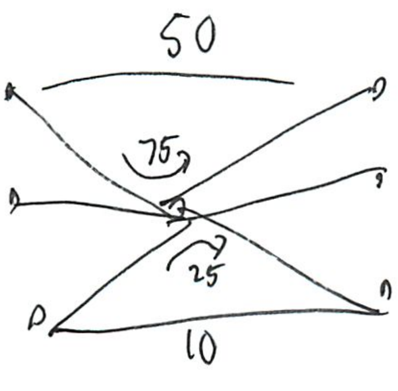
but must travel through capital

Sounds like The Hunger Games...

but if prob $>$ m that carted through Capital → then don't send there

if no path w/ prob $<$ m don't send

I don't get...



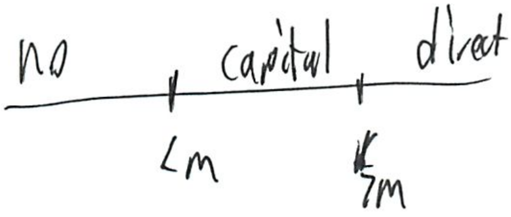
So $m = 50$

then 75 → no good through capital

So take 50 path

doesn't make logical sense why...

(15)



but not that simple → diff routes

First

1. Look at capital route
2. if >m → look at direct route(s)
   <m → capital route   ? not best one
3. if not <m >m → not sent
   if <m → do local route

But we still need prob for each path normal

And through capital $O(n^2)$

Then decide $O(1)$ for each step

(16)

(a) 29 says ~~modify~~ pretty much that but how to modify all pairs shortest path algorithm?

Basically I did that at end.

---

how get capital cate

a → cap    cap → b

~~(16)~~ Read Wiki

10/5
10/20

(17) b) We only run $1 \rightarrow$ ~~[illegible]~~ $\frac{1}{q}$ times

Assume only integer %

If has q

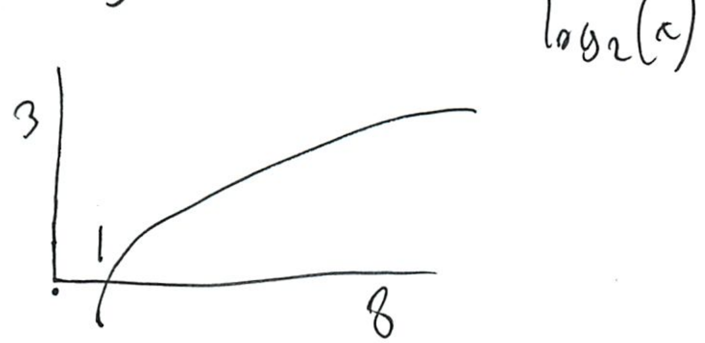
Oh q at each hop!

So 5% each hop → 20 hops

---

c) Why log p ?

log (x) is



but p is < 1

; just a way to reverse

- leave mine

| 6.046 Homework | **Problem Set 2** | Problem 1 |
|---|---|---|
| Yihui Saw | | October 5, 2012 |
| Collaborators: Baiaboo Rai | | |

Answer to problem 1

We will first represent the set of $n$ distinct numbers $S = \{a_1, a_2, ..., a_n\}$ as a polynomial $x^{a_1}+x^{a_2}+...+x^{a_n}$. Then multiplying the polynomials $x^{a_1}+x^{a_2}+...+x^{a_n}$ and $x^{a_1}+x^{a_2}+...+x^{a_n}$, we get the polynomial

$$\sum_{i=1}^{n} x^{a_i} + \sum_{1 \leq i < j \leq n} 2x^{a_i+a_j}$$

Notice that the power of the terms in the latter sum $a_i + a_j$ contains all possible sums of any pair of numbers in the set $S$. If we then compare all the powers in the latter terms and check if any power is equal to $a_i$, $a_i - 1$ or $a_i + 1$ for some $i$ then we are done.

To multiply the two polynomials, we use the Fast Fourier Transform to convert the polynomials into point value representation which takes $O(n \log n)$ time, then multiplication and addition takes $O(n)$ time and converting back to coefficient form takes $O(n \log n)$ time (as we have learned in lecture). Thus, multiplying the two polynomials takes $O(n \log n)$ time.

Now we argue that checking all sums also takes $O(n \log n)$ time. We can first sort the set $S$ in $O(n \log n)$ time using merge sort. Then the sum of any pair in $S$ is in the range $[1, 10n + 10n - 1]$. So there are $O(n)$ distinct sums and we can use a binary search on the sorted set, to check if any sum is equal to $a_i$, $a_i - 1$ or $a_i + 1$ for some $i$ in the sorted set. Thus, this takes $O(n \log n)$ time as well.

Hence, we can find whether there is some triple of three distinct elements $(a_i, a_j, a_k)$ such that $a_i = a_j + a_k$, $a_i = a_j + a_k + 1$, or $a_i = a_j + a_k - 1$ by multiplying the polynomials and comparing the sums in a total of $O(n \log n)$ time.

totally diff

6.046 Homework | **Problem Set 2** | Problem 2
Yihui Saw | | October 5, 2012
Collaborators: Baiaboo Rai

Answer to problem 2

a) If we chose to have a deterministic strategy and the producer finds out about the strategy, the producer can walk through the same steps of our deterministic strategy. The producer can then replace the terminating solution with the briefcase containing the least amount of money and we will always finish with the smallest reward.

b) Consider the following randomized strategy: We will open $\frac{n}{2}$ briefcases at random so the probability of choosing any briefcase is $\frac{1}{n}$. Call the set of opened briefcases $O$. We will then continue to open briefcases until we reach a briefcase such that the amount of money in the briefcase is greater than $max(O)$ or until there are no briefcases left. There are 4 cases that could occur:

1) The second best briefcase and the best briefcase are in set $O$. ) Can 4 occur if you have the max

2) The second best briefcase and the best briefcase are NOT in set $O$.
whats special about 2nd best?

3) The second best briefcase is NOT in set $O$ but the best briefcase is in set $O$.

4) The second best briefcase is in set $O$ but the best briefcase is NOT in set $O$.

We will lose in case 1, 2, 3 and get the best briefcase in case 4. Since the probability of choosing any briefcase is equal, we have a $\frac{1}{4}$ probability of getting the best briefcase.

We have a different solution from part (a) because we chose the briefcases at random. Without knowing which briefcases we will choose for the first $\frac{n}{2}$ briefcases, the producer will not be able to arrange the order of the briefcases in a way that would prevent case 4 from occuring.

again diff

6.046 Homework | **Problem Set 2** | Problem 3
Yihui Saw | | October 5, 2012
Collaborators: Baiaboo Rai

a) We will show that $\gcd(x+y,N)$ and $\gcd(x-y,N)$ are non-trivial divisors of $N$ by contradiction.

First, suppose there exist $x, y$ such that $\gcd(x+y,N)= N$. This implies that $x+y$ is a multiple of $N$ so

Not 1
N

$$x+y \equiv 0 \mod N \implies x \equiv -y \mod N$$

There is a contradiction. Similarly if $\gcd(x-y,N)= N$, then

$$x-y \equiv 0 \mod N \implies x \equiv y \mod N$$

which is also a contradiction by our assumption in the problem.

Not
1

Now, suppose there exist $x, y$ such that $\boxed{\gcd(x+y,N)= 1}$ i.e. $x+y$ and $N$ are relatively prime. But given $x^2 \equiv y^2 \mod N$, we have that

$$x^2 - y^2 \equiv 0 \mod N$$

which implies that

$$\overset{\cancel{0}}{(x+y)}\overset{0}{(x-y)} \equiv 0 \mod N$$

One of them is 0

But if $x+y$ is relatively prime to $N$ then it must be that $(x-y) \equiv 0 \mod N$. Then, $x \equiv y \mod N$ which is a contradiction. Similarly, if $\gcd(x-y,N)= 1$, then $x-y$ and $N$ are relatively prime and we have that $(x+y) \equiv 0 \mod N$ which is again a contradiction. Hence, $\gcd(x+y,N)$ and $\gcd(x-y,N)$ cannot be 1 or $N$ and so are non-trivial divisors of $N$.

both sides

b)

c)

d)

between any two cities on this graph represents the path with the smallest probability of the caravan being destroyed.

6.046 Homework | **Problem Set 2** | Problem 4
Yihui Saw | | October 5, 2012
Collaborators: Baiaboo Rai

a) We have $D$ hackies and each road costs $c$ so we can build $\frac{D}{c}$ number of roads. To maximize our monthly tax revenue, we want to maximize the number of cities we can reach from the capital with $\frac{D}{c}$ roads. With each road built, we can gain access to at most one city, so to maximize the number of cities we can reach, we can built roads in this way: continuously build roads from the capital to a new city connecting $(Hackerville, city_i)$ until we run out of money, building $\lfloor\frac{D}{c}\rfloor$ and gaining a taxation revenue of $p\lfloor\frac{D}{c}\rfloor$ per month.

didn't explain how

b) We will apply dynamic programming to find if a merchant can be sent between two cities and arrive with some stock left while travelling on the path that takes the shortest time. Notice that a merchant sells of $q$ percent of his original stock at each hop, so if a merchant still has some stock left to sell at his destination, the path from the source to the destination must have taken less than $\frac{1}{q} = q^{-1}$ hops. Thus, we will examine each city one at a time as a source node $s$ and ~~minimizing the time~~ it takes to get to other cities by considering paths to all other cities $d_i$ with $k < q^{-1}$ hops via this recurrence relation:

how is that equiv

Oh 70% means less than 70 hops, ? integer % only ?

$$DP(s, d_i, k) = \min_{u \in V}(DP(s, u, k-1) + c(u, d_i))$$

where $c(u, d_i)$ is the cost function defined by:

Dynamic programming

$$c(u, d_i) = \begin{cases} t_{u,d_i} & \text{if } (u, d_i) \text{ is an edge i.e. } (u, d_i) \in E \\ \infty & \text{otherwise} \end{cases}$$

$DP(s, d_i, k)$ refers to the minimum time it takes to travel from city $s$ to city $d_i$ with at most $k$ hops in the path. So if $DP(s, d_i, q^{-1})$ is not infinity then there is a shortest path between $s$ and $d_i$ where merchants can travel on and arrive with some stock left at $d_i$. The base case for the DP is the $DP(s, d_i, 0) = 0$. There are $n$ different nodes, each with $n$ different destinations, and we examine each step 1 to $q^{-1}$ $n$ times so the runtime of the algorithm is $n \cdot (n) \cdot (n) \cdot q^{-1} = O(n^3 q^{-1})$. In class we learned another method using matrix multiplication to represent the shortest path problem. There are no negative weight cycles so we can apply the algorithm with repeated squaring, so there are $\log(q^{-1})$ multiplications giving a run time of $O(n^3 \log(q^{-1})) = o(n^3 q^{-1})$.

I did not have → add

c)We want to minimize the probability a caravan will get destoryed while travelling between any two city. Notice that if a caravan was travelling from city $i$ to $j$ then to $k$, the probability of the caravan being destroyed is $p_{i,j} \cdot p_{j,k}$. Taking the log of this probability we get $\log p_{i,j} + \log p_{j,k}$. So if we let $\log p_{i,j}$ be the weight of the edge (the road with $p_{i,j}$), the shortest path

? why the log p ?

Michael Plasmeier

6.046 R07

X not submitted

P-Set 2

#1

---

Find a triple $(x, y, z)$ s.t. $z = x + y$

$z = x + y + 1$

$z = x + y - 1$

So we want to find two items that add together to form a third

~~Example 2 set~~

~~So we start in the~~

Example set 1, 2, 3, 4, 5, 6

We first sort the set $O(n \lg n)$

Then we start in the middle (ie 3, 4) and add them and ± 1 (ie 6, 7, 8) $O(3)$

We look these up in a hash table $O(3)$ if they are there, we return $x, y, z$

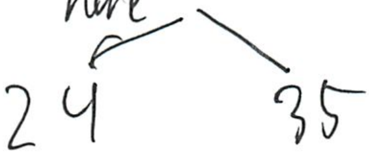
7

Not submitted

If not, we randomly move either the left ~~[illegible]~~ slot left / right slot right

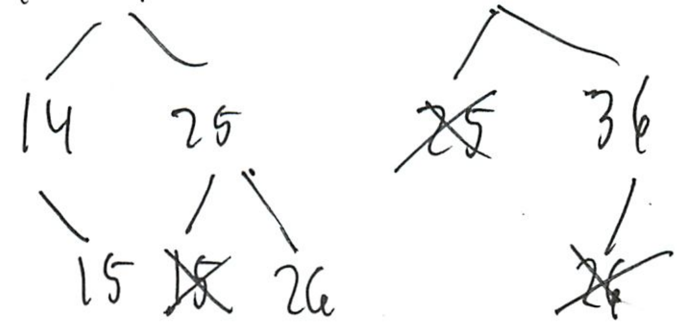
i.e. what was 3 to the left / 4 to the right

So we here



O(1)

We compare these as before
And continue to compare and memoize, so no duplicates



This grows $2T(\frac{n}{2}) + O(1)$

② But just a little bit of randomness could (in the producers strategy

b) Randomized strategy

⊗ not submitted

We don't know the <u>range</u> of the cases ~~at~~ in advance.

We can only see what is good based on opening several cases.

1st case

$P($1st case opened is largest$) = \frac{1}{N}$

since we know Nothing about the others

2nd case

For the second case, it can either be smaller than case 1 → in which case we know w/ certainty it is not the largest. Or it can be larger.

We have a 50-50 shot either way.

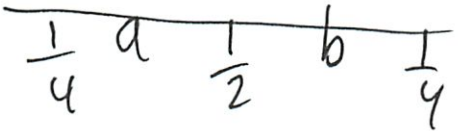
$\frac{1}{n/2} = \frac{2}{n}$ shot

3

Not submitted

3rd case

The 3rd case has a



$\frac{1}{2}$ shot of being < ~~a~~ Case 1

$\frac{1}{2}$ shot of being < case 2 ) ind

So a $\frac{1}{4}$ shot < case 1 and 2

like wise a $\frac{1}{4}$ shot < case 1
> Case 2

$\frac{1}{4}$ shot $\overset{>}{<}$ case 1
case 2 ) So $\frac{1}{2}$ chance in middle

$\frac{1}{4}$ shot > case 1
> case 2 ] only interested in for largest

So now ~~it~~ it is largest w/ prob $\frac{1}{4}$

or $\frac{1}{n/4} = \frac{4}{n}$

n cases

We see a ~~b~~ pattern

$$\frac{2^{r}}{n}$$

(3)(4)

not submitted

We set this equal or less then to $\frac{1}{4}$

$$\frac{2^r}{n} \leq \frac{1}{4}$$

And solve for r, given n

$$r = \frac{\log\left(\frac{n}{4}\right)}{\log(2)}$$

gives us after r rounds when we ~~are~~ have a $\geq \frac{1}{4}$ shot of having the largest

Michael Plasmeier

6.046 R07

P-Set 2

# 1

w/ Yihui

## Set Elements

We need to add up each pair of #s we could do this niavely ~~[illegible]~~ in $n^2$

~~[illegible]~~ Could we do better?

We can go back to lecture 3 w/ FFT.

In lecture we learned about how we can represent numbers as polynomials

$$S = \{ a_1, a_2, \cdots a_n \}$$

$$\downarrow$$

$$x^{a_1} + x^{a_2} + \ldots + x^{a_n}$$

When we multiply it by itself we get

$$\sum_{i=1}^{n} x^{a_i} + \underbrace{\sum_{1 \le i < j < n} 2\, x^{a_i + a_j}}_{\text{Contains all possible sums in the set}}$$

②

We do the multiplying w/ the FFT $O(n \lg n)$

multiply + add $O(n)$

convert back $O(n \lg n)$

We compare all powers in the later terms and check if any power is equal to $a_i$, $a_i - 1$, $a_{i+1}$

We sort the set $S$ $O(n \lg n)$ w/ merge sort

Then the sum of any pair in $S$ in in the range $[1, 10n + 10n - 1]$

So $\theta(n)$ distinct sums

Find w/ binary search → n times → $O(n \lg n)$ to see if something = to $a_i$, $a_i - 1$, $a_i + 1$

③

So if a triple of 3 distinct elements $(a_i, a_j, a_k)$ s.t.

$$a_i = a_j + a_k$$
$$a_i = a_j + a_k + 1$$
$$a_i = a_j + a_k - 1$$

Then return it!

$O(1)$

---

$O(n \lg n)$
time

Michael Plasmeier
6.046 R07

P-Set 2 #2

Yuki, pwoods

---

## Briefcase Game

Note: used following assumptions:

The values in each case were different each round and we don't know the possible values as in Deal or No deal

### a) Deteministic

After many rounds of a deteministic Strategy, the producer can "stack the deck" such that we never pick the highest case and instead pick the lowest.

---

But w/ just a little bit of randomess we could ruin producers strategy

②

b) Random strategy

Randomly group half of the cases into group A and B

~~Randomly~~

Open all of Group A

Now open one from B at a time till ~~B~~ b larger than max(A)

Then we have 4 possibilites for the largest and second largest.

| | | largest | |
|---|---|---|---|
| | | A | B |
| 2nd largest | A | 1 | 2 |
| | B | 3 | 4 |

Each has a 50% a-priori prob. to be in group A or B, ind.

So each box is Prob $= \frac{1}{4}$

(3)

Case 1| Both the largest and second largest were in the group we open. We lose.

Case 2| We saw the 2nd largest in A, so that was max(A). This means the true largest will be our decision b. We take it and we win.

Case 3| The largest box was already opened. We won't find any thing larger than max(A). We lose.

Case 4| Both the largest and 2nd largest have not been opened. We don't know about the 3rd largest, but there is some largest.. This means there are $\geq 2$ cases which $>$ max(A). We have no guarentee which one we pick - we might win

4

So we have a $\geq \frac{1}{4}$ shot of winning with this strategy

Michael Plasmeier
6.046 R07

P-Set 2 #3
w/ Yihui Saw

a) Show $\gcd(x+y, N)$ and $\gcd(x-y, N)$ are non tivial divisors of N by contidiction

↓

non tilvial means $\neq 1, N$
lets show each seperatly

not N

First suppose x, y exist such that
$\gcd(x+y, N) = N$

This must mean that $x+y$ is a multiple of N
That would mean that

$$x+y = 0 \mod N$$
$$x = -y \mod N$$

Which we told can't be true. Contidiction

We can also try $\gcd(x-y, N) = N$

$$X - y \equiv 0 \mod N$$

$$x = y \mod N$$

Another thing we are told can't be true,
Contridiction

---

can't be 1

Now suppose we had an x, y s.t.

$$\gcd(x+y, N) = 1$$

Which would mean x+y, N are rel. prime

But if we have

$$X^2 = y^2 \mod N$$

$$X^2 - y^2 = 0 \mod N$$

$$(x+y)(x-y) = 0 \mod N$$

If the (x+y) term is rel prime to N
then (x-y) term must = 0 mod N
but that would be x = y mod N, which
is a contidiction!

Likewise if $(x-y)$ term is rel prime to $N$ the $(x+y)$ term would be $\equiv 0 \bmod N$

But then $x \equiv -y \bmod N$

Another contradiction

Thus $\gcd(x+y, N)$ and $\gcd(x-y, N)$ can't be either $1$ or $N$ so must be non-trivial divisors of $N$. ∎

(4)

b) Las-Vegas ~~Primality~~ Non-Trivial Divisor Test

There are many Monte Carlo prob tests that are very likly, but not guarenteed to retrn a correct ans.

However, we want a Las Vegas algorithm which always returns the right thing in finite time - but uses randomness to do that process faster. For example QuickSort runs faster when its randomized in the avg case.

---

We should test for the easiest stuff first. If a # is prime, we can return "prime". We test using Fact 2. $O(n^k)$

(5)

Next we can see if N is even $O(1)$

Next can check if $N = a^b$ for integers $a, b > 1$

This is the check for perfect power $O(n^k)$

So now we know $x^2 = k \mod N$ has $0$ or $> 4$ sols

So we pick an integer $\{1, 2, \ldots, N-1\}$ at random $O(1)$

Then check if $\gcd(x, N) > 1$ $O(\log n)$

L if so we return $\gcd(x, N)$

Now we check $a = x^2 \mod N$ $O(1)$

$y = \text{SQRT}(a, N)$ $O(1)$

If $y = \pm x \mod N$

We pick another #  $O(1)$

and repeat this process

6

Otherwise we know gcd$(x + y, N)$ will be a non-trivial divisor for $N$, so we return it!

---

All this runs in $O(n^k)$ time

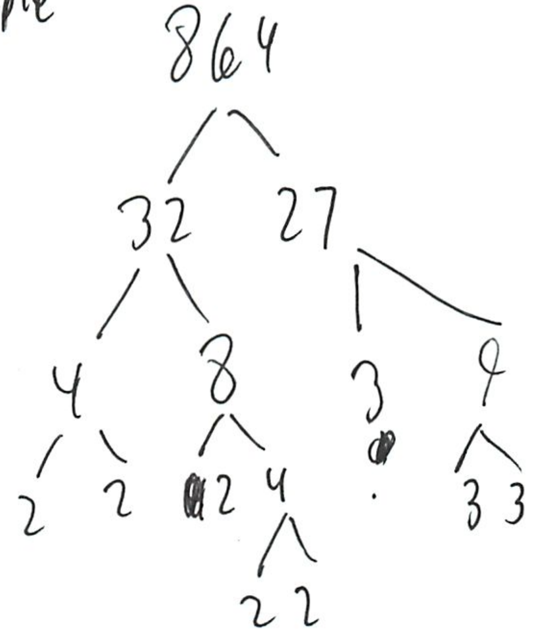
Avg time will be much lower because often we can return at the start

7

c) Prime Factorization

We can factor recursivly by using our One-Factor() function from b)

For example



So we basically have

```
Factor(N) :
        d = One Factor(N)
        if d is prime    ← fact 2
            └ return d
        else return Factor(d), Factor($\frac{d}{N}$)
```

⑧

Running Time

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n^k)$$

$n^{\lg_2 2}$ $n^k$

↑ larger

$O(n^k)$

This works due to the fund. theorm of arithmetic that every positive integer has a prime factorization.

d) SQRT Real?

We can dream!

Seriously → it's an imaginary function that does not exist in real life

Michael Plasmeier
6.046 R07

P-Set 2
#4
Yihui, pwoods

a) UKH Road System

I am assuming connected w/ capital means any connection - not just a direct connection

This is a Minimum spanning tree problem
But We need to look at the profit of each city

It is a MST problem, since we want to connect as many vertices as possible for as much $\Sigma p$

But we must also conside construction cost, We want to minimize $\Sigma c$ so that ve can connect as many cites as possible before hitting D (ar capital limit)

So for each <u>edge</u> - calculate the profit

$\pi = c - p$

↑profit in Course 14 speak

- yes p is a property of the vertex

2

We will only look at one month's ~~profit~~ revenue since ~~profit~~ revenue is effectivly static in this problem, and you asked us to look at monthly revenues.

Because we want to maximize revenue while minimizing cost, we want to maximize profit

We can a simple MST on these new profit weights, except we try to maximize.

The locally optimal solution is still best as the remaining problem will have same form

③

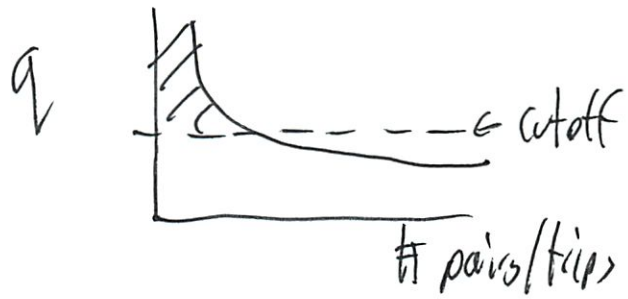
b) Greedy merchants

The Floyd Warsall algorithm can be used to find the minimum travel times between all pairs of all nodes

However we have the additional Constraint of trying to minimize q.

However q is a cutoff

Basically



So we run F-W → $O(n^3)$

Then we check what the remaining q is $O(\overset{n}{\cancel{q}})$
to get min travel times

On our given pair - by running it

If above our cutoff → return yes
below → return no

$$\frac{O(1)}{O(n^3)}$$

(3b)

Since $q$ is the % sold at each hop we can only go $\frac{1}{q}$ hops before we are sold out

ie 5% $\rightarrow \frac{1}{20}$ so only 20 hops till sold out

So we only need to do F-W for that # of hops, thus $O(n^3 q^{-1})$

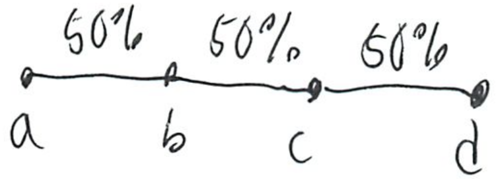
Through matrix multiplication we can do better $O(n^3 \lg(q^{-1}))$ which is $o(n^3 q^{-1})$

(4)

## c) Attacked by Bandits

The probability that a route is disrupted is 1-(the product of ~~the~~ each road segment being disrupted).

For example, each seg has an ind 50% chance it is disrupted.



$$\text{So the } p(\text{route disrupted}) = 1 - \left(\tfrac{1}{2} \cdot \tfrac{1}{2} \cdot \tfrac{1}{2}\right)$$
$$= 1 - \tfrac{1}{8}$$
$$= 7/8$$

So we again do F-W, but instead of adding paths, we multiply $O(n^3)$

5

d) Capital Transit:

To start at with, I used my all-pairs lowest probability table from c. $O(n^3)$

Then I look at the best rote from source s to capital and capital to t. a destination from our table $O(1)$ lookup

I then ~~compare~~ se if s,c + c,t is $< m$.

If it is → return s,c + c,t $O(1)$

If not, go back to our all pairs lowest prob and look there $O(1)$

If this is $< m$, ~~[illegible]~~ return this rote as before

If this is $> m$, then don't send caravan

$O(n^3)$

*Design and Analysis of Algorithms* October 5, 2010
Massachusetts Institute of Technology 6.046J/18.410J
Profs. Srini Devadas and Ronitt Rubinfeld Handout 8

# Problem Set 2 Solutions

This problem set is due **at 11:59pm** on **Friday, October 5, 2012.**

**Exercise 2-1.** Do Exercise 5.3-2 in CLRS.

**Exercise 2-2.** Do Exercise 7.3-2 in CLRS.

**Exercise 2-3.** Do Exercise 23.1-5 in CLRS.

**Exercise 2-4.** Do Exercise 23.1-10 in CLRS.

**Exercise 2-5.** Do Exercise 23.2-4 in CLRS.

**Exercise 2-6.** Do Exercise 23.2-8 in CLRS.

**Exercise 2-7.** Do Exercise 25.1-9 in CLRS.

**Exercise 2-8.** Do Exercise 25.3-6 in CLRS.

### Problem 2-1. Set Elements

Let $A$ be a finite set with $n$ distinct integer elements in the range of $[1, 10n]$. Give an efficient algorithm that runs in time $O(n \log n)$ to find whether there is some triple of three distinct elements $(x, y, z)$ such that $z = x + y$, $z = x + y + 1$, or $z = x + y - 1$.

**Solution:** This problem is similar to the Cartesian Sum problem in CLRS. Create a polynomial $P(x)$ from the set:

$$P(x) \quad = a_1x^1 + a_2x^2 + ...a_nx^n$$

where $a_i = 1$ if element $i \in A$. Square the polynomial, which can be done efficiently using FFT, to obtain a new polynomial

$$\begin{aligned} P(x)^2 &= Q(x) \\ &= q_0x^{b_0} + q_1x^{b_1} + ... + q_{2n}x^{b_n} \end{aligned}$$

The values $b_k$ will be exactly the values of $a_i + a_j$.

This resulting polynomial's coefficients $q_k$ will be non-zero positive integers if there exists a pair of elements $i, j \in A$ such that $k = i + j$. The resulting polynomial will have up to $20n$ elements, but we only need to check the first $10n$. This can be done by hashing the resulting values, and then iterating over the original set to test for inclusion. The same process is used to check for the $+1$ and $-1$ offset triples. Care must be taken, however, to make sure that we do not include those terms composed of elements for which $i = j$, since this represents a triple $(x, x, z)$. This can be done by checking if an element $\frac{b_k}{2} \in A$. If so, we check $q_k$. If $q_k = 1$, then $b_k$ was a result of $i = j$ and we disregard this element. Otherwise, $b_k$ was the sum of two non-identical integers and so represents a valid triple.

**Problem 2-2. Briefcase Game**

Consider the following gameshow concept, inspired by "Deal or No Deal." You are given a table with $n$ identical briefcases, each with a cash prize inside of an unknown amount. The rules of the game are as follows. You are allowed to open one briefcase at a time and look at how much money is inside. At that point, you must choose whether to keep this cash prize or move on to the next briefcase. Once you reject a briefcase, you may not claim its cash anymore. Your goal is to come up with a strategy to choose the briefcase with the largest cash prize. A strategy is an algorithm which attempts to maximize some profit in the presence of an adversary, which in turn is an algorithm that attempts to minimize your profit. In this case, the game show producer is your adversary.

**(a)** Show that a deterministic strategy for this gameshow is terrible. That is, show if you use a deterministic algorithm then a gameshow producer that happens to know your algorithm can always trick you into choosing the briefcase with the least amount of money.

**Solution:** A deterministic algorithm will always open the briefcases in some particular order. That is, when opening briefcase $i$ the contestant makes a decision of whether to keep it or not based off the last $i - 1$ briefcases, and makes a deterministic decision of which briefcase to open next. A computationally unbounded adversary (i.e. gameshow producer) will be able to look down the algorithm's decision tree, and always make the relative rank of briefcase $i$ worse than the previous $i - 1$ briefcases. Therefore, the contestant will always be led down the decision tree along the worst possible path.

**(b)** Show instead that with a randomized strategy you will be able to choose the best briefcase with a probability of $\frac{1}{4}$. Why do you get a different result from part (a)? *Optional*: Can you do better?

**Solution:** The randomized strategy avoids the conundrum of part (a) by randomizing which briefcases are opened. The adversary will therefore not be able to lead you down the worst possible path of the decision tree.

Choose a random permutation $\pi(n)$ representing the order in which the briefcases are opened. Look at the contents of the first $\frac{n}{2}$ briefcases and reject them. Let $v$ be the largest cash prize seen after opening these briefcases. Look at the next $\frac{n}{2}$ briefcases and accept the first briefcase that is better than $v$ or the last briefcase (if we have run out of briefcases). We will get the best cash prize if the second-best briefcase is in the first $\frac{n}{2}$ briefcases and the best briefcase lies in the second $\frac{n}{2}$ briefcases. The total probability is therefore $(\frac{1}{2})(\frac{1}{2}) = \frac{1}{4}$, as desired.

A better solution can be achieved by rejecting the first $\frac{n}{e}$ briefcases and accepting the next best one. This leads to a probability of $\frac{1}{e}$ of accepting the best cash prize. This is actually a well-known optimization problem known as the best prize problem.

**Problem 2-3. Square Roots Mod N**

Let $N$ be a positive integer. We will be dealing with arithmetic mod $N$ in this problem. In particular, we are interested in multiplication and taking square roots mod N. This is much like taking square roots in the real numbers, except that the roots must be integers. For example, if $N = 5$, then $\sqrt{4} = \{2, 3\}$, since $2^2 = 3^2 = 4 \bmod 5$. However, not all numbers are squares (for example, neither 2 nor 3 are squares for $N = 5$).

First, some terminology. Define the set $Z_N = \{0, 1, \ldots, N-1\}$ that represents an arithmetic set where (1) integers may be multiplied mod N and (2) multiplication is associative. A special property is that if $\gcd(a, N) = 1$ for all $a \in Z_N$ then all elements of $a$ have inverses, i.e. there exists an element $a^{-1}$ such that $a^{-1}a = 1 \bmod N$ (sanity check: for what numbers $N$ does this occur?). Next, call $Z_N^* = \{a \in Z_N \mid \gcd(a, N) = 1\}$. In this set all elements have inverses, and it is closed under multiplication. $Z_N^*$ is called the multiplicative group of integers modulo N. Finally, remember that a composite number is a non-prime number (i.e. it can be factored).

Assume we have a black box for computing square roots modulo N; that is, a deterministic algorithm $\text{SQRT}(a, N)$ that, given positive integers $N$ and $a \in [1, 2, \ldots N]$, returns in $O(1)$ time an integer $x$ satisfying $a \equiv x^2 \bmod N$ if such an $x$ exists and 0 otherwise. In this problem we will use SQRT to give an expected polynomial time randomized algorithm for factoring integers.

We will also need the following facts:

**Fact 1:** If $N > 1$ is an odd composite integer that is not a perfect power (i.e., $N \neq a^b$ for any integers $a, b > 1$), then for any $k \in Z_N^*$, the equation $x^2 \equiv k \bmod N$ has either zero or at least four distinct solutions in $Z_N^*$.

**Fact 2:** Primality testing can be performed in deterministic polynomial time.

**(a)** We say that $d$ is a *non-trivial divisor* of $N$ if $d$ divides $N$ and $d$ is neither 1 nor $N$. Suppose $N$ is a composite integer. For integers $x, y$, show that if $x^2 \equiv y^2 \bmod N$ but $x \not\equiv y \bmod N$ and $x \not\equiv -y \bmod N$, then $\gcd(x + y, N)$ and $\gcd(x - y, N)$ are non-trivial divisors of $N$.

**Solution:** First we prove the following lemma: if $a, b$ are integers such that $a, b \not\equiv 0 \bmod N$ but $ab \equiv 0 \bmod N$, then $\gcd(a, N)$ and $\gcd(b, N)$ are non-trivial divisors of $N$.

Suppose, to get a contradiction, that $\gcd(a, N) = 1$. Then $a \in Z_N^*$ and has a multiplicative inverse $a^{-1}$. Multiplying both sides of $ab \equiv 0 \bmod N$ by $a^{-1}$ we get $b \equiv 0 \bmod N$, contrary to given assumption. Thus, $\gcd(a, N) \neq 1$ (or $N$, since $a \not\equiv 0 \bmod N$). By symmetry the same holds for $b$.

Returning to the main problem, since $x^2 \equiv y^2 \bmod N$, we can write

$$0 \equiv x^2 - y^2 \equiv (x + y)(x - y) \bmod N.$$

But since $y \not\equiv \pm x \bmod N$, we know $x \pm y \not\equiv 0 \bmod N$. Applying our lemma completes the proof.

**(b)** Give a Las Vegas algorithm that, given an integer $N > 1$, outputs "prime" if $N$ is prime and, if $N$ is composite, outputs a non-trivial divisor of $N$. Your algorithm should make use of SQRT and run in expected polynomial time. (Hint: Use the facts above, and beware of perfect powers.)

**Solution:**

```
ONE-FACTOR($N$)
1  if $N$ is prime, return "prime"
2  if $N$ is even, return 2
3  if $N = a^b$ for some integers $a, b > 1$, return $a$
4  Pick $x$ from $\{1, 2, \ldots, N-1\}$ uniformly at random
5  if $\gcd(x, N) > 1$, return $\gcd(x, N)$
6  Let $a := x^2 \bmod N$
7  Let $y :=$ SQRT($a, N$)
8  if either $y \equiv x \bmod N$ or $y \equiv -x \bmod N$, goto Line 4
9  else return $\gcd(x + y, N)$
```

If $N$ is prime, by Line 1 will catch it (courtesy of Fact 2). Suppose instead that $N$ is composite. By Fact 1, there are at least four values SQRT$(a, N)$ could return in Line 7 (we checked for $N$ being even or a perfect power, and there has to be at least one solution, namely, $x$). Since it is deterministic, it will always return $y$. But our $x$ was random, so $x$ was just as likely to have been one of the two (or more) that were *not* $\pm y \bmod N$. In other words, there is a probability of at least $1/2$ that the algorithm will go to Line 9. In this case, part (a) tells us that we can safely return either $\gcd(x + y, N)$ or $\gcd(x - y, N)$. If this failed, we simply repeat from Line 4, with the same probability of success ($\geq 1/2$) on each independent trial.

Each step in ONE-FACTOR runs in time polynomial in $\log N$ (the number of bits in the input), including checking if $N$ is a perfect power in Line 3: we can check each value of $b$, $2 \le b \le \log_2 N$, by looking for a matching $a$ via binary search (since having fixed $b$, $f(a) := a^b$ is a monotone function) within the range $2 \le a \le N$. Moreover, each time we execute Lines 4-8 we have a probability of at least $1/2$ of not having to repeat, so the expected running time of ONE-FACTOR is at most twice the running time of a single trial, therefore polynomial as well.

**(c)** Give a Las Vegas algorithm that, given an integer $N > 1$, outputs a list of all prime factors of $N$ (with multiplicities, and in any order). Your algorithm should still run in expected polynomial time.

**Solution:**

```
FACTOR($N$)
Compute $d :=$ ONE-FACTOR($N$)
if $d =$ "prime", return $d$
else return FACTOR($d$), FACTOR($N/d$)
```

This procedure clearly outputs the desired factorization of $N$. To analyze its running time, consider the recursion tree it traverses. This binary tree has as many leaves as $N$ has prime factors (with multiplicities), which is at most $\log_2 N$, so it also has at most $\log_2 N$ internal nodes. Each node or leaf represents one call to ONE-FACTOR, which takes expected polynomial time, so FACTOR also runs in polynomial time.

**(d)** Factoring integers in polynomial time (expected or deterministic) is a longstanding open problem. What does this say about SQRT?

**Solution:** It is at least as hard as factoring. In other words, it doesn't exist, as far as we know.

### Problem 2-4. Merchant Tycoon

You have been named minister of trade in the middle ages in the distant United Hackers Kingdom (UHK). The UHK has a set of $n$ major cities each with goods it wishes to sell and buy. Your job is to manage the trade network and trade caravans of the kingdom in order to increase revenues.

**(a)** Your first assignment is to improve the road system of the UKH. There are currently no roads, and you are given $D$ hackies (the units of currency) to build them. Each city with a trade route to the capital, Hackerville, will create a taxation revenue of $p$ hackies per month. Constructing a road between two cities $(i, j)$ has a constant cost $c$. Create an efficient algorithm that maximizes the monthly revenue from trade routes.

**Solution:** We are looking for an MST that costs less than $D$ weight. We can run Prim's algorithm with the capital as the starting node. However, because all the costs are the same, then there is actually the much simpler solution of adding $\frac{D}{c}$ roads from the capital to the other cities. This in fact only takes $O(E)$ time, which is faster than Prim's algorithm.

**(b)** With tax revenue flowing in, you have managed to connect all your cities to each other with roads. The time to travel between cities $(i, j)$ is $t_{i,j}$. Your next task is to manage the trade caravans of the kingdoms. You would ideally like to send one caravan between every two cities in your kingdom so that goods from one city will appear in every other city. Merchants are greedy, however. Each time a merchant visits a city he will sell off at most $q$ percent of his original stock in the black market, such that by the time the merchant reaches the destination he may have no more stock left to sell. Devise an efficient algorithm that will find if a merchant can be sent between two cities and arrive with some stock left, and if so it will do so in the fastest time possible. Your algorithm should run in $o(n^3 q^{-1})$.

**Solution:** Run the matrix multiplication shortest path algorithm but only carry out the multiplication up to $q'$ times, where $q'$ is the largest integer that is no greater than $\frac{1}{q}$. This gives a performance of $O(n^3 \log q')$ with repeated squaring of matrices.

**(c)** Terrible news! An attack by bandit raiders has made the roads unsafe to travel, which has greatly affected the revenue influx from your trade network. On the plus side, the caravans are no longer corrupt. Each road has a probability $p_{i,j}$ that the caravan will be destroyed. You would still like to send caravans between every two cities to keep trade alive. Give an efficient algorithm that gives the paths between cities where the probability that the caravans will be raided is minimized.

**Solution:** First, turn the probabilities that the caravans are destroyed $p_{i,j}$ and turn it into the probability that the caravans will not be destroyed $p'_{i,j} = 1 - p_{i,j}$. Now the probability that a caravan gets to its destination safely is the product of the probabilities along the paths. Use the log probability of the these new weights and take their negatives. Since multiplying probabilities is equivalent to adding negative log probabilities, the weights $-\log p'_{i,j}$ may be used with any all pairs shortest paths to find the safest path .

**(d)** The raids have caused major food shortages throughout the kingdom. Each city produces one staple that must be distributed to every other city. The king has decided that the capital is more important than the other cities, and prefers that caravans travel through them to keep its food stocks full. However, if the caravan has a probability of more than $m$ of getting raided while traveling through the capital then it is preferable to not send the caravan through there. If no path with overall probability less than $m$ is found, then the caravan should not be sent at all. Redesign your algorithm from part (c) to take this new constraint into account.

**Solution:** Run Floyd-Warshall, using the same weights from part (c) and putting the node corresponding to the capital city in the front of the nodes list. If a path with weight less than $-\log m$ is found going through the capital, keep this path. If not, test the shortest path for weight, and if it is still greater than or equal to $-\log m$ then do not send out a caravan. The other solution is to run Johnson's algorithm, but in each iteration of Dijkstra to check the path through the capital first and then check for the shortest path. Either solution will work.

**Michael E Plasmeier**

| | |
|---|---|
| **From:** | Aizana Turmukhametova <aizana@MIT.EDU> |
| **Sent:** | Wednesday, October 03, 2012 2:29 PM |
| **To:** | Aizana Z Turmukhametova |
| **Subject:** | 6.046/18.410 announcement: Quiz 1 Announcement (must read) |
| **Follow Up Flag:** | Flag for follow up |
| **Flag Status:** | Flagged |

*Note: This mail was sent to all students in the stellar class Design and Analysis of Algorithms*

# Quiz 1 Announcement (must read)

**What:** Quiz 1

**When:** Thursday, October 11, 2012, 11:05am - 12:25 pm during normal class hours. The exam is 80 minutes long. Please show up early so that we can start and finish on time.

**Where:** 26-100 (normal lecture room). Please leave a free seat next to you on both sides unless you're sitting next to the aisle. Please fill the room from front to back and try to seat efficiently.

**Special Accommodation:** If you require a special accommodation for the quiz such as a distraction free room and extra time, please let us know ASAP. You should email aizana@mit.edu before Thursday, October 4.

**Crib sheet:** The quiz is closed book. Books, notes, laptops, calculators, phones, etc. are not allowed. You may, however, bring one crib sheet on an 8 1/2" x 11" or A4 paper. You may use both sides of the paper and write or print as small as you like. You are allowed only one crib sheet and no other helper material or device.

**Material covered:** You are responsible for all material covered up to October 5: Lectures 1-9, Recitations 1-4, Problem Sets 1-2. You are also responsible for the material covered in the corresponding sections in the textbook, as indicated by the reading assignments on the course calendar.

**Quiz format:** There will be true/false questions, and short and long problems that are in total worth 80 points. The quiz format is similar to previous years, but the material coverage may differ. A sample quiz from the previous term has been posted.

**Quiz review:** During recitation 4 on Friday, October 5. Additional quiz review OH will be scheduled.

This announcement was made in Stellar on 2012 October 03 by Aizana Turmukhametova

The announcement is also posted on the class website:
https://stellar.mit.edu/S/course/6/fa12/6.046J/index.html

how do you study for this class?

*Design and Analysis of Algorithms*
Massachusetts Institute of Technology
Profs. Dana Moshkovitz and Bruce Tidor

Pratice 10/10

March 6, 2012
6.046J/18.410J
Quiz 1

# Quiz 1

- Do not open this quiz booklet until you are directed to do so. Read all the instructions first.
- The quiz contains 4 problems, several with multiple parts. You have 80 minutes to earn 80 points.
- This quiz booklet contains 10 pages, including this one, and a sheet of scratch paper which can be detached.
- This quiz is closed book. You may use one double-sided letter ($8\frac{1}{2}'' \times 11''$) or A4 crib sheet. No calculators or programmable devices are permitted. Cell phones must be put away.
- Write your solutions in the space provided. If you run out of space, continue your answer on the back of the same sheet and make a notation on the front of the sheet.
- Do not waste time deriving facts that we have studied. It is sufficient to cite results from class.
- When we ask you to "give an algorithm" in this quiz, describe your algorithm in English or pseudocode, and provide a short argument for correctness and running time. You do not need to provide a diagram or example, unless it helps make your explanation clearer.
- Do not spend too much time on any one problem. Generally, a problem's point value is an indication of how many minutes to spend on it.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Please be neat.
- Good luck!

| Problem | Title | Points | Parts | Grade | Initials |
|---|---|---|---|---|---|
| 0 | Name | 1 | 1 | | |
| 1 | True or False | 24 | 8 | | |
| 2 | Translation | 25 | 5 | | |
| 3 | All Pairs Shortest Red/Blue Paths | 18 | 3 | | |
| 4 | Telephone Psetting | 12 | 3 | | |
| Total | | 80 | | | |

**Name:** ____________________

Circle your recitation:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| R09 | R10 | R03 | R04 | R07 | R08 | R05 | R06 |
| F10 | F11 | F12 | F1 | F1 | F2 | F2 | F3 |
| Yuri | Yuri | Igor | Igor | Sarah | Sarah | Lin | Lin |

**Problem 0. Name.** [1 point] Write your name on every page of this exam booklet! Don't forget the cover.

**Problem 1. True or False.** [24 points] (8 parts)

Circle **T** or **F** for each of the following statements to indicate whether the statement is true or false, and briefly explain why. Your justification is worth more points than your true-or-false designation.

**(a)** **T** [✓ **F**] The solution to the recurrence $T(n) = 2^n T(n-1)$ is $T(n) = \Theta((\sqrt{2})^{n^2+n})$. (Assume $T(n) = 1$ for $n$ smaller than some constant $c$).

Master [illegible] can't

each level $2^n$

and each gets one smalle

$n(2^n)$

There ans seems wrong

**(b)** [✓ **T**] **F** The solution to the recurrence $T(n) = T(n/6) + T(7n/9) + O(n)$ is $O(n)$. (Assume $T(n) = 1$ for $n$ smaller than some constant $c$). (i two

$n^{\lg_9 1}$

$9^x = 1$ right

So $O(n)$ bigger

$O(n)$

Alg qs are like consulting qs - I can almost ans them

**(c)** [✓ **T**] **F** In a simple, undirected, connected, weighted graph with at least three vertices and unique edge weights, the heaviest edge in the graph is in no minimum spanning tree.

Would not know how to prove

? what does connected mean - every line

Could run alg and show it always picks least

**(d)** **T** [✓ **F**] The weighted task scheduling problem with weights in the set $\{1, 2\}$ can be solved optimally by the same greedy algorithm used for the unweighted case.

? What

forgot - need to review

does not sound defensible

**(e)** [✓ **T**] **F** Two polynomials $p$, $q$ of degree at most $n - 1$ are given by their coefficients, and a number $x$ is given. Then one can compute the multiplication $p(x) \cdot q(x)$ in time $O(\log n)$.

That special thing we learned

L review

ah forgot these — no less than ?

**(f)** T [✓ F] Suppose we are given an array $A$ of $n$ distinct elements, and we want to find $n/2$ elements in the array whose median is also the median of $A$. Any algorithm that does this must take $\Omega(n \log n)$ time.

Don't see it

But no proof otherwise?

locating O(1)?

**(g)** T F There is a density $0 < \rho < 1$ such that the asymptotic running time of the Floyd-Warshall algorithm on graphs $G = (V, E)$ where $|E| = \rho |V|^2$ is better than that of Johnson's algorithm. (review alg

**(h)** T [✓ F] Consider the all pairs shortest paths problem where there are also weights on the vertices, and the weight of a path is the sum of the weights on the edges and vertices on the path. Then, the following algorithm finds the weights of the shortest paths between all pairs in the graph:

```
APSP-WITH-WEIGHTED-VERTICES$(G, w)$:
for $(u, v) \in E$
    Set $w'(u, v) = (w(u) + w(v))/2 + w(u, v)$
Run Johnson's algorithm on $G, w'$ to compute the distances $\delta'(u, v)$ for all $u, v \in V$.
for $u, v \in V$
    Set $d_{uv} = \delta'(u, v) + \frac{1}{2}(w(u) + w(v))$
```

(why make everything $\frac{1}{2}$ makes no sense

**Problem 2. Translation** [25 points] (5 parts)

You have been hired to manage the translation process for some documentation. Unfortunately, different sections of the documentation were written in different languages: $n$ languages in total. Your boss wants the entire documentation to be available in all $n$ languages.

There are $m$ different translators for hire. Some of those translators are volunteers that do not get any money for their services. Each translator knows *exactly* two different languages and can translate back and forth between them. Each translator has a non-negative hiring cost (some may work for free). Unfortunately, your budget is too small to hire one translator for each pair of languages. Instead, you must rely on chains of translators: an English-Spanish translator and a Spanish-French translator, working together, can translate between English and French. Your goal is to find a minimum-cost set of translators that will let you translate between every pair of languages.

hiring/recruiting cost → not hourly!

We may formulate this problem as a connected undirected graph $G = (V, E)$ with non-negative (i.e., zero or positive) edge weights $w$. The vertices $V$ are the languages for which you wish to generate translations. The edges $E$ are the translators. The edge weight $w(e)$ for a translator $e$ gives the cost for hiring the translator $w(e)$. A subset $S \subseteq E$ of translators can be used to translate between $a, b \in V$ if and only if the subgraph $G_S = (V, S)$ contains a path between $a$ and $b$. The set $S \subseteq E$ is a translation network if and only if $S$ can be used to translate between all pairs $a, b \in V$.

**(a)** Prove that each minimum spanning tree of $G$ is also a minimum-cost translation network.

Know min spanning tree alg

Since our only cost is a Fixed, pre determined hiring/recruiting cost - the parameters of our problem match up w/ min spanning tree

We are seeking the min cost way b/w two points which is the ~~[illegible]~~ shortest path along min spanning tree.

Actually, not really - may not be optimum for only 2

**(b)** Give an example of a minimum-cost translation network that is not a minimum spanning tree of $G$.

ah - not what I wrote earlie
thought it was oter way so trouble
thinking of something

**(c)** Give an efficient algorithm that takes $G$ as input, and outputs a minimum-cost translation network of $G$. State the runtime of your algorithm in terms of the number of languages $n$ and the number of potential translators $m$.

Min spaning tree
(guess not seeing above)

Your bosses have decided that the previous approach to translation doesn't work. When attempting to translate between Spanish and Portuguese — two relatively similar languages — it degrades the translation quality to translate from Spanish to Tagalog to Mandarin to Portuguese. There are certain clusters of languages that are more closely related than others. When translating between two languages that lie within the same cluster, such as Spanish and Portuguese, the translation is of high quality when the sequence of languages used to translate between them is completely contained within the cluster.

More formally, the language set $V$ can be divided into disjoint clusters $C_1, \ldots, C_k$. Each cluster $C_i$ contains languages that are fairly similar; each language is contained in exactly one cluster. Your bosses have decided that a translation between $a, b \in C_i$ is high-quality if and only if all of the languages used on the path from $a$ to $b$ are also in $C_i$. The translator set $S$ is a high-quality translation network if and only if it is a translation network, and for any language cluster $C_i$ and any languages $a, b \in C_i$, $S$ can be used for a high-quality translation between $a$ and $b$.

Clusta of networks — HQ inside whole network

AND

**(d)** Suppose that $S$ is a minimum-cost high-quality translation network. Let $S_i = S \cap (C_i \times C_i)$ be the part of the network $S$ that lies within the cluster $C_i$. Show that $S_i$ is a minimum-cost translation network for the cluster $C_i$.

What is that notation?
So S is what we look at
and Ci are our network
Since S is min cost hq
and made up of Cis → then so

**(e)** Give an efficient algorithm for computing a minimum-cost high-quality translation network. Analyze the runtime of your algorithm in terms of the number of languages $n$ and the number of translators $m$.

Compute the networks b/w nodes?
Well hq is only within node
when outside node → lq?

**Problem 3. All Pairs Shortest Red/Blue Paths.** [18 points] (3 parts)

You are given a directed graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}$. In addition, each edge of the graph is either red or blue. The shortest red/blue path from vertex $i \in V$ to vertex $j \in V$ is defined as the shortest path from $i$ to $j$ among those paths that go through *exactly* one red edge (if there are no such paths, the length of the shortest red/blue path is $\infty$).

We can represent this graph with two $n \times n$ matrices of edge weights, $W_r$ and $W_b$, where $W_r$ contains the weights of all red edges, and $W_b$ contains the weights of all blue edges.

why like that?

**(a)** Given the Floyd-Warshall algorithm below, how would you modify the algorithm to obtain the lengths of the shortest paths that only go through blue edges?

Just use $W_b$ instead of $W$

```
FLOYD-WARSHALL(W):
n = W.rows
D^(0) = W
for k = 1 to n
    let D^(k) = (d_ij^(k)) be a new n × n matrix
    for i = 1 to n
        for j = 1 to n
            d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
return D^(n)
```

**(b)** How would you modify your algorithm from part (a) to keep track not only of shortest paths with only blue edges, but also those with exactly one red edge, and to output the lengths of the shortest red/blue paths for all pairs of vertices in this graph?

( augment data structure to track # of red edges

( but is then what we have still optimal

**(c)** Prove the correctness of your algorithm using a loop invariant.

Proof start

Proove transition no change

**Problem 4. Telephone Psetting.** [12 points] (3 parts)

Upon realizing that it was 8:30 PM on Wednesday and he had not yet started his 6.046 pset, Ben Bitdiddle found $n - 1$ other students (for a total of $n$ students) in the same situation, and they decided to do the pset, which coincidentally had $n$ problems in total, together.

Their brilliant plan for finishing the pset in time was to sit in a circle and assign one problem to each student, so that for $i = 0, 1, \ldots, n-1$, student $i$ did problem number $i$, and wrote up a solution for problem $i$ meriting $p(i)$ points. Then, they each copied the solutions to all the other problems from the student next to them, so that student $i$ was copying from student $i - 1$ (and student 0 was copying from student $n - 1$).

lol

Unfortunately, they were in such a hurry that the copying chain degraded the quality of the solutions: by the time student $i$'s solution to problem $i$ reached student $j$, where $d = j - i \ (mod\ n)$, $d \in \{0, 1, \ldots, n - 1\}$, the solution was only worth $\frac{1}{d+1}p(i)$ points.

**(a)** Write a formula that describes the total pset score $S(x)$ for student $x$, where the total pset score is the sum of the scores that student $x$ got on each of the $n$ problems.

Write a recurrence $\Sigma$

**(b)** Describe a simple $O(n^2)$ algorithm to calculate the pset scores of all the students.

run it

**(c)** Describe a $O(n \log n)$ algorithm to calculate the pset scores of all the students.

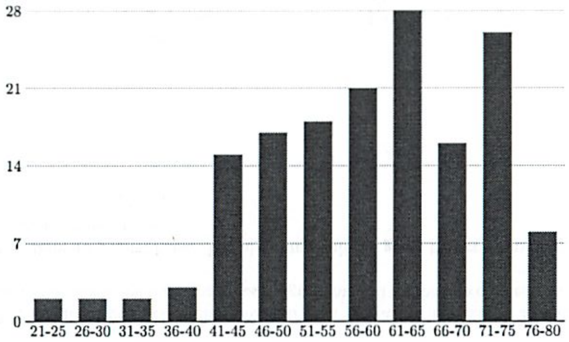
do something clever - DP

memozation?

SCRATCH PAPER

*Design and Analysis of Algorithms* | March 6, 2012
Massachusetts Institute of Technology | 6.046J/18.410J
Profs. Dana Moshkovitz and Bruce Tidor | Quiz 1 Solutions

# Quiz 1 Solutions



Mean: 58; Median: 60; Standard deviation: 12

**Problem 0. Name.** [1 point] Write your name on every page of this exam booklet! Don't forget the cover.

**Problem 1. True or False.** [24 points] (8 parts)

Circle **T** or **F** for each of the following statements to indicate whether the statement is true or false, and briefly explain why. Your justification is worth more points than your true-or-false designation.

**(a) T F** The solution to the recurrence $T(n) = 2^n T(n-1)$ is $T(n) = \Theta((\sqrt{2})^{n^2+n})$. (Assume $T(n) = 1$ for $n$ smaller than some constant $c$).

**Solution:** [3 points] True. Let $T(0) = 1$.
Then $T(n) = 2^n \cdot 2^{n-1} \cdot 2^{n-2} \ldots 2^1 = 2^{(n+(n-1)+(n-2)+\ldots+1)}$.
Because $\sum_{i=1}^{n} i = n(n+1)/2$, we therefore have $T(n) = 2^{(n^2+n)/2} = (\sqrt{2})^{n^2+n}$.
Some students also correctly solved the problem by using the substitution method. Some students made the mistake of multiplying the exponents instead of adding them. Also, it has to be noted that $2^{n^2/2} \neq \Theta(2^{n^2})$.

**(b) T F** The solution to the recurrence $T(n) = T(n/6) + T(7n/9) + O(n)$ is $O(n)$. (Assume $T(n) = 1$ for $n$ smaller than some constant $c$).

**Solution:** [3 points] True. Using the substitution method:

$$\begin{aligned} T(n) &\leq cn/6 + 7cn/9 + an \\ &\leq 17cn/18 + an \\ &\leq cn - (cn/18 - an) \end{aligned}$$

This holds if $c/18 - a \geq 0$, so it holds for any constant $c$ such that $c \geq 18a$.
Full credit was also given for solutions that uses a recursion tree, noting that the total work at level $i$ is $(17/18)^i n$, which onverges to $O(n)$.

(c) **T F** In a simple, undirected, connected, weighted graph with at least three vertices and unique edge weights, the heaviest edge in the graph is in no minimum spanning tree.

**Solution:** [3 points] False. If the heaviest edge in the graph is the only edge connecting some vertex to the rest of the graph, then it must be in every minimum spanning tree.

(d) **T F** The weighted task scheduling problem with weights in the set $\{1, 2\}$ can be solved optimally by the same greedy algorithm used for the unweighted case.

**Solution:** [3 points] False. The algorithm will fail given the set of tasks (given in the form $((s_i, f_i), w_i)$:
$\{((0, 1), 1), ((0, 2), 2)\}$.

(e) **T F** Two polynomials $p, q$ of degree at most $n - 1$ are given by their coefficients, and a number $x$ is given. Then one can compute the multiplication $p(x) \cdot q(x)$ in time $O(\log n)$.

**Solution:** [3 points] False. We need at least $\Theta(n)$ time to evaluate each polynomial on $x$ and to multiply the results. Some students argued incorrectly that it must take $O(n \log n)$ using FFT, but FFT overkills because it computes all coefficients, not just one.

(f) **T F** Suppose we are given an array $A$ of $n$ distinct elements, and we want to find $n/2$ elements in the array whose median is also the median of $A$. Any algorithm that does this must take $\Omega(n \log n)$ time.

**Solution:** [3 points] False. It's possible to do this in linear time using SELECT: first find the median of $A$ in $\Theta(n)$ time, and then partition $A$ around its median. Then we can take $n/4$ elements from either side to get a total of $n/2$ elements in $A$ whose median is also the median of $A$.

(g) **T F** There is a density $0 < \rho < 1$ such that the asymptotic running time of the Floyd-Warshall algorithm on graphs $G = (V, E)$ where $|E| = \rho |V|^2$ is better than that of Johnson's algorithm.

**Solution:** [3 points] False. The asymptotic running time of Floyd-Warshall is $O(V^3)$, which is at best the same asymptotic running time of Johnson's (which runs in $O(VE + V \log V)$ time), since $E = O(V^2)$

(h) **T F** Consider the all pairs shortest paths problem where there are also weights on the vertices, and the weight of a path is the sum of the weights on the edges and vertices on the path. Then, the following algorithm finds the weights of the shortest paths between all pairs in the graph:

```
APSP-WITH-WEIGHTED-VERTICES$(G, w)$:
for $(u, v) \in E$
    Set $w'(u, v) = (w(u) + w(v))/2 + w(u, v)$
Run Johnson's algorithm on $G, w'$ to compute the distances $\delta'(u, v)$ for all $u, v \in V$.
for $u, v \in V$
    Set $d_{uv} = \delta'(u, v) + \frac{1}{2}(w(u) + w(v))$
```

**Solution:** [3 points] True. Any shortest path from $u$ to $v$ in the original graph is still a shortest path in the new graph. For some path $\{v_0, v_1, \ldots, v_k\}$, we have:

$$\begin{aligned}
w'(v_0 \rightsquigarrow v_k) &= \sum_{i=0}^{k-1}((w(v_i) + w(v_{i+1}))/2 + w(v_i, v_{i+1})) \\
&= \sum_{i=0}^{k-1} w(v_i, v_{i+1}) + \frac{1}{2}\left(\sum_{i=0}^{k-1} w(v_i) + \sum_{i=1}^{k} w(v_i)\right) \\
&= \sum_{i=0}^{k-1} w(v_i, v_{i+1}) + \sum_{i=0}^{k} w(v_i) - \frac{1}{2}(w(v_0) + w(v_k)) \\
&= w(v_0 \rightsquigarrow v_k) - \frac{1}{2}(w(v_0) + w(v_k))
\end{aligned}$$

Therefore, the order of all paths from $v_0$ to $v_k$ remains unchanged so Johnson's algorithm in line 3. finds the correct path, and the adjustment in line 5 finds the correct length $d_{v_0 v_k}$.

**Problem 2. Translation** [25 points] (5 parts)

You have been hired to manage the translation process for some documentation. Unfortunately, different sections of the documentation were written in different languages: $n$ languages in total. Your boss wants the entire documentation to be available in all $n$ languages.

There are $m$ different translators for hire. Some of those translators are volunteers that do not get any money for their services. Each translator knows *exactly* two different languages and can translate back and forth between them. Each translator has a non-negative hiring cost (some may work for free). Unfortunately, your budget is too small to hire one translator for each pair of languages. Instead, you must rely on chains of translators: an English-Spanish translator and a Spanish-French translator, working together, can translate between English and French. Your goal is to find a minimum-cost set of translators that will let you translate between every pair of languages.

We may formulate this problem as a connected undirected graph $G = (V, E)$ with non-negative (i.e., zero or positive) edge weights $w$. The vertices $V$ are the languages for which you wish to generate translations. The edges $E$ are the translators. The edge weight $w(e)$ for a translator $e$ gives the cost for hiring the translator $w(e)$. A subset $S \subseteq E$ of translators can be used to translate between $a, b \in V$ if and only if the subgraph $G_S = (V, S)$ contains a path between $a$ and $b$. The set $S \subseteq E$ is a translation network if and only if $S$ can be used to translate between all pairs $a, b \in V$.

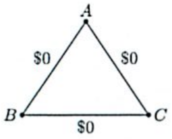
**(a)** Prove that each minimum spanning tree of $G$ is also a minimum-cost translation network.

**Solution:** [5 points] Let $T$ be some minimum spanning tree of $G$. Because $T$ is a spanning tree, there is a path in $T$ between any pair of vertices. Hence, $T$ is a translation network.

For the sake of contradiction, suppose that $T$ is not a minimum-cost translation network. Then there must be some translation network $S$ with total cost strictly smaller than $T$. Because $S$ connects every pair of vertices, it must have some spanning tree $T^*$ as a subgraph. Because all edge weights are nonnegative, we have $w(T^*) \leq w(S) < w(T)$. Hence, $T$ is not the minimum spanning tree. This contradiction means that all spanning trees are also minimum-cost translation networks.

**(b)** Give an example of a minimum-cost translation network that is not a minimum spanning tree of $G$.

**Solution:** [5 points] If the graph of translators contains a cycle of translators all willing to work for \$0, then it is possible to hire all of the translators in the cycle without increasing the overall cost of the translation network. The smallest example of this is the following:



All three MSTs of the graph have total cost \$0, so any translation network of cost \$0 has minimum cost. Hence, we can take all translators in the cycle to get a minimum-cost translation network that is not an MST.

**(c)** Give an efficient algorithm that takes $G$ as input, and outputs a minimum-cost translation network of $G$. State the runtime of your algorithm in terms of the number of languages $n$ and the number of potential translators $m$.

**Solution:** [5 points] We saw in part **(a)** that every minimum spanning tree is a minimum-cost translation network of $G$. Hence, to find a minimum-cost translation network, it is sufficient to find a minimum spanning tree of $G$. We may do so using Kruskal's algorithm, for a runtime of $\Theta(m \log n)$, or using Prim's algorithm, for a runtime of $\Theta(m + n \log n)$.

Your bosses have decided that the previous approach to translation doesn't work. When attempting to translate between Spanish and Portuguese — two relatively similar languages — it degrades the translation quality to translate from Spanish to Tagalog to Mandarin to Portuguese. There are certain clusters of languages that are more closely related than others. When translating between two languages that lie within the same cluster, such as Spanish and Portuguese, the translation is of high quality when the sequence of languages used to translate between them is completely contained within the cluster.

More formally, the language set $V$ can be divided into disjoint clusters $C_1, \ldots, C_k$. Each cluster $C_i$ contains languages that are fairly similar; each language is contained in exactly one cluster. Your bosses have decided that a translation between $a, b \in C_i$ is high-quality if and only if all of the languages used on the path from $a$ to $b$ are also in $C_i$. The translator set $S$ is a high-quality translation network if and only if it is a translation network, and for any language cluster $C_i$ and any languages $a, b \in C_i$, $S$ can be used for a high-quality translation between $a$ and $b$.

**(d)** Suppose that $S$ is a minimum-cost high-quality translation network. Let $S_i = S \cap (C_i \times C_i)$ be the part of the network $S$ that lies within the cluster $C_i$. Show that $S_i$ is a minimum-cost translation network for the cluster $C_i$.

**Solution:** [5 points] Let $S_i^*$ be a minimum-cost translation network for $C_i$. Because $S$ is a high-quality translation network, $S_i$ must contain a path between every pair of nodes in $C_i$, so $S_i$ is a translation network for $C_i$. For the sake of contradiction, assume that $S_i$ is not minimum-cost. Then $w(S_i) > w(S_i^*)$.
Consider the translation network $S^* = (S - S_i) \cup S_i^*$. Then $w(S^*) = w(S) - w(S_i) + w(S_i^*) < w(S)$. Because $S_i^*$ is a translation network of $C_i$, replacing $S_i$ with $S_i^*$ will not disconnect any pair of vertices in the graph. Furthermore, any pair of vertices connected by a path in $S$ that lay inside a particular cluster will be connected by a path in $S^*$ that lies within the same cluster. Hence, $S^*$ is a high-quality translation network with cost strictly less than $S$. This contradicts the definition of $S$, so $S_i$ must be a minimum-cost translation network.

**(e)** Give an efficient algorithm for computing a minimum-cost high-quality translation network. Analyze the runtime of your algorithm in terms of the number of languages $n$ and the number of translators $m$.

**Solution:** [5 points] The idea behind this algorithm is to first compute one MST for each individual cluster, and then to compute a global MST using the remaining edges. More specifically, we do the following:

1. For each edge $(u, v)$, if there is some cluster $C_i$ such that $u, v \in C_i$, then add $(u, v)$ to the set $E_i$. Otherwise, add $(u, v)$ to the set $E_{global}$.

2. For each cluster $C_i$, run Kruskal's algorithm on the graph $(C_i, E_i)$ to get a minimum-cost translation network $T_i$ for the cluster. Take the union of these minimum to get a forest $T$.
3. Construct an empty graph $G_{global}$ on nodes $\{1, \ldots, k\}$. For each edge $(u, v)$ in $E_{global}$, where $u \in C_i$ and $v \in C_j$, check whether the edge $(i, j)$ is in the graph $G_{global}$. If so, set $w(i, j) = \min\{w(i, j), w(u, v)\}$. Otherwise, add the edge $(i, j)$ to the graph $G_{global}$. In either case, keep a mapping $source(i, j) = (u^*, v^*)$ such that $w(i, j) = w(u^*, v^*)$.
4. Run Kruskal's algorithm on the graph $G_{global}$ to get $T_{global}$.
5. For each edge $(i, j) \in T_{global}$ add the edge $source(i, j)$ to $T$.

We begin by examining the runtime of this algorithm. The first step requires us to be able to efficiently discover the cluster $C_i$ that contains each vertex, which can be precomputed in time $\Theta(m)$ and stored with the vertices for efficient lookup. So this filtering step requires $\Theta(1)$ lookup per edge, for a total of $\Theta(m)$ time.

The second step is more complex. We run Kruskal's algorithm on each individual cluster. So for the cluster $C_i$, the runtime is $\Theta(|E_i| \lg |C_i!)$. The total runtime here is:

$$\sum_{i=1}^{k} a \cdot |E_i| \lg |C_i| \leq \sum_{i=1}^{k} a \cdot |E_i| \lg n = (a \lg n) \sum_{i=1}^{k} |E_i| \leq am \lg n$$

Hence, the total runtime for this step is $\Theta(m \lg n)$.

The third step also requires care. We can store the graph to allow us to efficiently lookup $w(\cdot, \cdot)$ and to tell whether an edge $(i, j)$ has been added to the graph. We can also store $source(\cdot, \cdot)$ to allow for $\Theta(1)$ lookups. So the runtime here is bounded by $\Theta(m)$. The fourth step is Kruskal's again, only once, on a graph with $\leq n$ vertices and $\leq m$ edges, so the total runtime is $\Theta(m \lg n)$. The final step involves a lookup for each edge $(i, j) \in T$, for a total runtime of $\Theta(k) \leq \Theta(n)$. Hence, the runtime is dominated by the two steps involving Kruskal. So the total worst-case runtime is $\Theta(m \lg n)$.

Next we consider the correctness of this algorithm. Suppose that the result of this process is not the minimum-cost high-quality translator network. Then there must be a high-quality translator network $S$ that has strictly smaller cost. Because $S$ is a minimum-cost high-quality translator network, we know that the portion of $S$ contained in the cluster $C_i$ is a minimum-cost translator network for $C_i$, which has the same cost as the minimum spanning tree for that cluster computed in step 2 of the algorithm. So the total weight of all inter-cluster edges in $S$ must be strictly less than the total weight of all inter-cluster edges in $T$. But the set of all inter-cluster edges in $T$ formed a minimum spanning tree on the cluster, so any strictly smaller set of edges cannot span the set of all clusters. So $S$ cannot be a translation network. This contradicts our assumption, and so $T$ must be a minimum-cost high-quality translation network.

**Problem 3. All Pairs Shortest Red/Blue Paths.** [18 points] (3 parts)

You are given a directed graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}$. In addition, each edge of the graph is either red or blue. The shortest red/blue path from vertex $i \in V$ to vertex $j \in V$ is defined as the shortest path from $i$ to $j$ among those paths that go through *exactly* one red edge (if there are no such paths, the length of the shortest red/blue path is $\infty$).

We can represent this graph with two $n \times n$ matrices of edge weights, $W_r$ and $W_b$, where $W_r$ contains the weights of all red edges, and $W_b$ contains the weights of all blue edges.

**(a)** Given the Floyd-Warshall algorithm below, how would you modify the algorithm to obtain the lengths of the shortest paths that only go through blue edges?

```
FLOYD-WARSHALL($W$):
1  $n = W.rows$
2  $D^{(0)} = W$
3  for $k = 1$ to $n$
4      let $D^{(k)} = (d_{ij}^{(k)})$ be a new $n \times n$ matrix
5      for $i = 1$ to $n$
6          for $j = 1$ to $n$
7              $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
8  return $D^{(n)}$
```

**Solution:** [6 points] In order to find shortest paths going through only blue edges, it suffices to ignore the red edges and run Floyd-Warshall on only the blue edges of the graph. We make the following changes:

- Replace each occurrence of $W$ with $W_b$ in lines 1 and 2.
- Replace the matrices $D^{(k)}$ with blue versions $D_b^{(k)}$ in lines 2, 4, and 8.
- Replace the matrix elements $d_{ij}^{(k)}$ with blue versions $d_{b,ij}^{(k)}$ in lines 4 and 7.

While the second and third changes above are unnecessary for this part, they lay the groundwork for future parts below.

**(b)** How would you modify your algorithm from part (a) to keep track not only of shortest paths with only blue edges, but also those with exactly one red edge, and to output the lengths of the shortest red/blue paths for all pairs of vertices in this graph?

**Solution:** [6 points] Add a new set of matrices $D_r^{(k)}$ that give lengths of shortest paths with exactly one red edge and intermediate vertices up to $k$. The resulting pseudocode is as follows:

```
RED-BLUE-FLOYD-WARSHALL($W_r, W_b$):
1   $n = W_r.rows$
2   $D_b^{(0)} = W_b$
3   $D_r^{(0)} = W_r$
4   for $k = 1$ to $n$
5       let $D_b^{(k)} = (d_{b,ij}^{(k)})$, $D_r^{(k)} = (d_{r,ij}^{(k)})$ be new $n \times n$ matrices
6       for $i = 1$ to $n$
7           for $j = 1$ to $n$
8               $d_{b,ij}^{(k)} = \min(d_{b,ij}^{(k-1)}, d_{b,ik}^{(k-1)} + d_{b,kj}^{(k-1)})$
9               $d_{r,ij}^{(k)} = \min(d_{r,ij}^{(k-1)}, d_{r,ik}^{(k-1)} + d_{b,kj}^{(k-1)}, d_{b,ik}^{(k-1)} + d_{r,kj}^{(k-1)})$
10  return $D_r^{(n)}$
```

**(c)** Prove the correctness of your algorithm using a loop invariant.

**Solution:** [6 points] The procedure for keeping track of paths that go through only blue edges is exactly equivalent to running Floyd-Warshall on the subgraph that contains only the blue edges of $G$, which is sufficient to show the correctness of $D_b^{(k)}$ for all $k$.

Loop invariant: at the end of every iteration of the for loop from $k = 1$ to $n$, we have both the length of the shortest path for every pair $(i, j)$ going through only blue edges and the length of the shortest path for every pair $(i, j)$ going through exactly one red edge, in each using intermediate vertices only up to $k$.

Initialization: At initialization $k = 0$, there are no intermediate vertices. The only blue paths are blue edges, and the only paths with exactly one red edge (red/blue paths) are red edges, by definition.

Maintenance: Each iteration gets the shortest blue-edges-only path going from $i$ to $j$ using intermediate vertices up through $k$ accurately, due to the correctness of the Floyd-Warshall algorithm.

For the paths including exactly one red edge, for a given pair $(i, j)$, there are two cases: either the shortest red/blue path from $i$ to $j$ using intermediate vertices through $k$ does not go through $k$, or it does. If it does not go through $k$, then the length of this path is equal to $d_{r,ij}^{(k-1)}$. If it does go through $k$, then the red edge on this path is either between $i$ and $k$ or between $k$ and $j$. Because of the optimal substructure of shortest paths, we can therefore break this case down into two subcases: the shortest path length is equal to $\min(d_{r,ik}^{(k-1)} + d_{b,kj}^{(k-1)}, d_{b,ik}^{(k-1)} + d_{r,kj}^{(k-1)})$. Line 9 in the algorithm above finds the minimum path length of these three different possibilities, so this iteration must find the shortest path length from $i$ to $j$ going through exactly one red edge, using only intermediate vertices through $k$.

Termination: After n passes through the loop, all paths with intermediate vertices up to $n$ have been included, and as there are only $n$ vertices, shortest paths using all vertices as intermediates will be discovered.

**Problem 4. Telephone Psetting.** [12 points] (3 parts)

Upon realizing that it was 8:30 PM on Wednesday and he had not yet started his 6.046 pset, Ben Bitdiddle found $n - 1$ other students (for a total of $n$ students) in the same situation, and they decided to do the pset, which coincidentally had $n$ problems in total, together.

Their brilliant plan for finishing the pset in time was to sit in a circle and assign one problem to each student, so that for $i = 0, 1, \ldots, n-1$, student $i$ did problem number $i$, and wrote up a solution for problem $i$ meriting $p(i)$ points. Then, they each copied the solutions to all the other problems from the student next to them, so that student $i$ was copying from student $i - 1$ (and student 0 was copying from student $n - 1$).

Unfortunately, they were in such a hurry that the copying chain degraded the quality of the solutions: by the time student $i$'s solution to problem $i$ reached student $j$, where $d = j - i \ (mod\ n)$, $d \in \{0, 1, \ldots, n - 1\}$, the solution was only worth $\frac{1}{d+1}p(i)$ points.

(a) Write a formula that describes the total pset score $S(x)$ for student $x$, where the total pset score is the sum of the scores that student $x$ got on each of the $n$ problems.

**Solution:** [3 points] $S(x) = \sum_{i=0}^{n-1} \frac{1}{((x-i) \mod n) + 1} \cdot p(i)$

Alternatively, it is also correct to give the equivalent sum:

$$S(x) = \sum_{i=0}^{n-1} \frac{1}{i+1} \cdot p((x-i) \mod n)$$

(b) Describe a simple $O(n^2)$ algorithm to calculate the pset scores of all the students.

**Solution:** [3 points] Use the formula given in part (a) to calculate each student's score. Because calculating the score requires summing $n$ numbers, it takes $O(n)$ time to calculate a single score, and therefore $O(n^2)$ time to calculate all the scores.

(c) Describe a $O(n \log n)$ algorithm to calculate the pset scores of all the students.

**Solution:** [6 points] The formula in the solution to part (a) describes a convolution: letting $g(y) = \frac{1}{y+1}$, the formula in part (a) can be written as $S : \mathbb{Z}_n \to \mathbb{R}$, where $S(x) = \sum_{i \in \mathbb{Z}_n} p(i)g(x-i) = (p \otimes g)(x)$. The algorithm is as follows:

1. Apply FFT on $p$ and $g$ to get $\hat{p}$ and $\hat{g}$ (time $\Theta(n \log n)$).
2. Compute the transformed convolution $\hat{S} = \hat{p} \cdot \hat{g}$ (time $\Theta(n)$).
3. Apply the inverse FFT to get back the convolution $S$ (time $\Theta(n \log n)$).

Alternatively, define two polynomials:

$P_1(x) = \sum_{i=0}^{2n-1} p(i \mod n)x^i$ and $P_2(x) = \sum_{i=0}^{n-1} \frac{1}{i+1} x^i$. Then student $j$'s pset score is equal to the coefficient of $x^{n+j}$ in the product of the polynomials $P_1$ and $P_2$. Because we are multiplying two polynomials of degree at most $2n - 1$, we can apply the polynomial multiplication method seen in class, which is outlined above: apply the FFT to get the evaluations of $P_1$ and $P_2$ on the roots of unity of order $2n$, pointwise multiply, and finally apply the inverse FFT to get the coefficients of the product.

SCRATCH PAPER

6.046 Exam Study

10/10
11p

## Interval Scheduling

$R = \{1, 2, \ldots, n)$

$s(i) < f(i)$

$f(j) \leq s(i)$

max size

So its ~~not~~ greedy

but by ends earliest

Prove w/ induction

---

Weighted

want the subset w/ the max weight

Greedy doesn't work

DP

(2)

Which is forward scan and backward scan
or whatever it is called
learned in 6.006

Try each request

$$\max(W_i + opt\ (R^{f(i)}))$$

$O(n^2)$

Did we ever say why?

Sort to ↓ complexity $O(n \lg n)$

---

non identical machines → actually impossible
NP-hard

3

1. Approx alg
   within some factor of optimal
   L = MonteCarlo?

Las Vegas → random time, but correct ans – randomized QuickSort

2. Heuristics to reduce problem size
   – bounding
   ILV

3. Greedy or suboptimal heuristic

4. Reduction to engines in Linear Programming

(4)

**Induction claim**

$m > k$

then $R_{jk+1}$ is not conflicting

$f(i_k) \leq f(j_k)$

So greedy would have added it

I think I know how to do this proving thing just out of practice

---

**Master Theorm**

do later

---

**Convex Hull**

n pts on plane

what is smallest polygon that includes all pts

⑤

a-la Mege sort

↳I forget which sort is which....

## Divide + Conquer

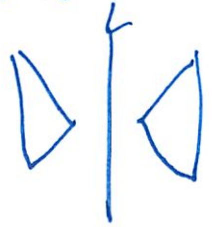
Divide in half

compute CH(left) CH(right)

Combine the 2 halves

↳ the fun part

Look highest intersection on L?



$n^2$

Or 2 finger alg

One moves clockwise or CCW

linear time

↳should I look at closer

6

## Median Finding

find the rank of a # within a list of #

rank(x) is #s in set $\leq x$

median rank $\left\lfloor \frac{n+1}{2} \right\rfloor$

Divide + Conquer

half + half

randomly pick x

but want x in middle

arrange in cols of size 5

$\lceil n/5 \rceil$ cols

Sort each col

Find median of medians

WP: based on Quicksort

BFRPT

7

5 Since that many registers

Choose median of medians as pivots

sort them

find median of

Then take all els to the right >x

at least $(3\lceil \frac{n}{5} \rceil - 2)$ els >x

$$T(n) = \begin{cases} O(1) \\ T(\lceil \frac{n}{5} \rceil) + T(\frac{7n}{10} + 6) \end{cases}$$

Ohh so we that group method to pick our pivot x

Then we continue on divide + conquer

k = rank(x)

?select on that group

8

# FFT

Adding polynomials of degree $< n$

ie. $7x^2 - 10x + 9$

$$\begin{array}{r} 7x^2 - 10x + 9 \\ +\ 4x - 5 \\ \hline 7x^2 - 6x + 4 \end{array}$$

takes $O(n)$ time

Multiply polynomicales $\rightarrow O(n^2)$

$\deg(C) = \deg(A) + \deg(B)$

Represent instead as the values at distinct inputs

$(x_0, y_0), (x_1, y_1), \ldots (x_{n-1}, y_{n-1})$

bij mapping b/w set of deg n polynomials

# POLYNOMIAL MULTIPLICATION AND THE FFT

## THE SCHOOL BOOK METHOD

Read 10/10

Given polynomial A(x) = $6x^3 + 7x^2 - 10x + 9$ and B(x) = $-2x^3 + 4x - 5$, their product C(x) can be calculated by the simple "school book" method as follows:

¿why called that

$$\begin{array}{rrrrrrr}
 & & & 6x^3 & +\ 7x^2 & -\ 10x & +\ 9 \\
 & & & -\ 2x^3 & & +\ 4x & -\ 5 \\
\hline
 & & & -\ 30x^3 & -\ 35x^2 & +\ 50x & -\ 45 \\
 & & 24x^4 & +\ 28x^3 & -\ 40x^2 & +\ 36x & \\
-\ 12x^6 & -\ 14x^5 & +\ 20x^4 & -\ 18x^3 & & & \\
\hline
-\ 12x^6 & -\ 14x^5 & +\ 44x^4 & -\ 20x^3 & -\ 75x^2 & +\ 86x & -\ 45
\end{array}$$

In summation form, if A(x) and B(x) are of degree $m$,

$$C(x) = \mathrm{Sum}_{j=0}^{2m}\ c_j\, x^j, \text{ where } c_j = \mathrm{Sum}_{k=0}^{j}\ a_k\, b_{j-k}.$$

¿how do we do evaluation -just do it I guess but which x values?

Polynomial A(x), above, could be shown in a *coefficient representation* as the vector of coefficients (9, -10, 7, 6). [Note that coefficients are typically specified with the $x^0$ coefficient first, then the coefficient of the $x^1$ term, and so on in increasing order.]

0 1 2 3

Alternatively, it could be specified in a *point-value representation* by evaluating it at $m$+1 distinct points. For example, if A(x) were evaluated at the points $x$ = 0, 1, 3, -1, its point-value representation would be {(0, 9), (1, 12), (3, 204), (-1, 20)}.

Why these?

The inverse of evaluation is *interpolation*. That is, the coefficient representation of a polynomial can be derived from a point-value representation by interpolation. Any set of $m$+1 point-value pairs ($x_i$, $y_i$) such that all the $x_i$ values are distinct uniquely defines a polynomial.

interpolation = going backwards

If two polynomials are specified in point-value representation using the same evaluation points, they can be multiplied by pointwise multiplication. However, because the product C(x) of two $m$-degree polynomials will be of degree 2$m$, we would need to extend the point-value representation of polynomials A(x) and B(x) to 2$m$+1 points in order to be able to interpolate C(x) from the pointwise multiplication of the 2$m$+1 points of A(x) and B(x).

For example, if A(x) and B(x) above are each evaluated at the points $x$ = -3, -2, -1, 0, 1, 2, and

3, their point-value representations would be

A = {(-3, -60), (-2, 9), (-1, 20), (0, 9), (1, 12), (2, 65), (3, 204)}

B = {(-3, 37), (-2, 3), (-1, -7), (0, -5), (1, -3), (2, -13), (3, -47)}

Since the product C(x) can be found by pointwise multiplication of two polynomials in point-value representation, polynomial multiplication in this form is O($n$), compared with the O($n^2$) cost of the "school book" multiplication of polynomials in coefficient representation.

## Convolution

The *convolution* of two $n$-vectors U and V, denoted U * V, is an $n$-vector W with components

$$w_i = \sum_{j=0}^{n-1} u_j \, v_{i-j},$$

where 0 <= $i$ < $n$ and the indices on the right-hand side are taken modulo $n$. The convolution of two vectors is like the multiplication of two polynomials in coefficient form where the results are wrapped around and added. If the coefficient representations of two $m$-degree polynomials are extended by padding the representation with $m$ zero coefficients as placeholders for the higher-order terms, then polynomial multiplication is equivalent to convolution.

## The Discrete Fourier Transform

The discrete Fourier transform (DFT) of a vector $v = (v_0, v_1, \ldots, v_{n-1})$ is a vector $y = (y_0, y_1, \ldots, y_{n-1})$ where

$$y_k = \sum_{j=0}^{n-1} v_j \, \mathit{omega}_n^{kj},$$

where $\mathit{omega}_n = e^{2\pi i/n}$ is the *principal nth root of unity*, a complex number such that $\mathit{omega}_n^k = 1$ for $k$ = 0, 1, ..., $n$-1. The value of $\mathit{omega}_n$ and the other complex $n$th roots of unity (the powers of $\mathit{omega}_n$) can be calculated using the definition of the exponential of a complex number:

$$e^{iu} = \cos(u) + i \sin(u).$$

If an $n$-vector $v$ is the coefficient representation of a polynomial, then the DFT of $v$ is equivalent to the evaluation of the polynomial at the $n$ complex $n$th roots of unity. That is, the DFT can be used to convert a polynomial from coefficient representation to point-value representation by evaluation at $n$ points.

The inverse DFT, denoted as $\mathrm{DFT}^{-1}$, performs the reverse conversion, the interpolation from a point-value representation to a coefficient representation:

$$a_j = 1/n \ \underset{k=0}{\overset{n-1}{\mathrm{Sum}}} \ y_k \ omega_n^{-kj},$$

for $j = 0, 1, ..., n-1$.

## The Convolution Theorem

For any two $n$-vectors $a$ and $b$, where $n$ is a power of 2,

$$a * b = \mathrm{DFT}_{2n}^{-1}(\mathrm{DFT}_{2n}(a) \cdot \mathrm{DFT}_{2n}(b))$$

where the vectors $a$ and $b$ are padded with leading 0's to length $2n$, and where * denotes convolution and · denotes the componentwise product of two $2n$-element vectors.

This means that polynomial multiplication can be accomplished using the DFT, the inverse DFT, and the pairwise multiplication of vectors.

## THE FAST FOURIER TRANSFORM

Using the fact that, if $n > 0$ is even, the squares of the $n$ complex $n$th roots of unity are the $n/2$ complex $(n/2)$th roots of unity, the DFT can be computed in $O(n \log n)$ time instead of $O(n^2)$ time, using a divide-and-conquer strategy: the even-index and odd-index coefficients of $A(x)$ are used separately to define the two polynomials $A^{even}(x)$ and $A^{odd}(x)$ of size $n/2$. This method, (re)discovered by Cooley and Tukey in the 1960's, is called the fast Fourier transform (FFT).

Since both the FFT and the inverse FFT run in time $O(n \log n)$ and pairwise multiplication of vectors is $O(n)$, using the FFT to perform convolution results in a $O(n \log n)$ algorithm for polynomial multiplication.

I don't exactly see this...

interpolation = going backward

I never got the Fourier Transform

but I think I see evaluation

and multiplication pointwise

(will likely not have to do - just know $O(1)$)

Add point wise $A(x) + B(x) = (A+B)(x)$

Multiply point value $A(x) \circ B(x) = (A \circ B)(x)$

Though need $2n$ values

So use a redundant representation

10

Coeff form

↓ eval

point value form

↓ multiply

point value

↓ interpolation

Coeff form

I totally get this

Eval

eval at each pt is n

↳ so $O(n^2)$

Can pick any distinct points

Simulareas computation → FFT

Ohhh!

So chose n pts

$\pm x_0, x_1, \ldots x_{\frac{n}{2}-1}$

11

$$A_{even}(z) = a_0 + a_2 z + a_4 z^2 + a_6 z^3$$

$$A_{odd}(z) = a_1 + a_3 z + a_5 z^2 + a_7 z^3$$

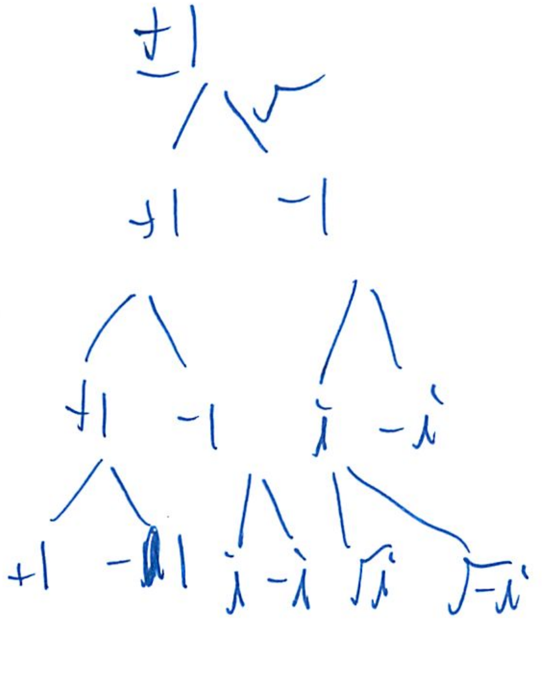
$$A(x_i) = A_{even}(x_i^2) + x_i A_{odd}(x_i^2)$$

$$A(-x_i) = A_{even}(x_i^2) - x_i A_{odd}(x_i^2)$$

Something about complex #s

(this class more about the app of this)

<u>Complex roots of unity</u>



(2)

$$e^{iu} = \cos(u) + i\sin(u)$$

Sol to $w^n = 1$ are $e^{2\pi i k/n}$

for $k = 0, 1, \ldots, n-1$

$i = \sqrt{-1}$

1. nth roots are ± paired

$$W_n^j = -W^{n/2+j} = e^{\frac{2\pi i(n/2+i)}{n}}$$

$$= \underbrace{\left(e^{2\pi i/2}\right)}_{-1}\left(e^{2\pi i j/n}\right)$$

$$= -W_n^{n/2+j}$$

2. Squaring the nth root gives $\frac{n}{2}$ th roots

⑬

Wp: Ahh so FFT is for continuous ↔ discrete

Or eqn to point representation and back again

Did in 6.02

↳ was the hardest lab

and 18.06

Why does it still not feel natural?

Interpolation

a special case of matrix multiply

14

## Randomized Alg

aka probabilistic

randomized quick sort

Can't find upper bound on a regular recurrence

$\log_{4/3} 2cn \cdot 2cn$

↑ # of levels ↑ size of each level

↑ i just that analysis

## Skip lists

randomized data structure

like express train in NYC

14 ⟶ 50 ⟶ 79

14 → 34 → 50 → 66 → 79

14 23 34 42 50 59 66 72 79

expected time for each

$$\boxed{EV = \frac{1}{prob}}$$

weighted avg of all possibilites

## Check Math w/ Randomized

Matrix multiplication verification

$$\boxed{A \times B \overset{?}{=} C}$$

$B \cdot r$

$A(B \cdot r)$

$C \cdot r$

are $A \cdot B \cdot r = C \cdot r$ ?

but some vectors don't work

### Freivalds' Alg

not going to look into details
seems pretty rare

I remember something about picking a random $n \times 1$ vector of $\{0,1\}$

Polynomial identites

Check factoring? well multiplication

Could have up to n terms

Which x do we need to try?

either 0 or at most n roots

So eval f at n+1 distinct locations

So pick x randomly

What should I put on the cheat sheet?

Algorithms

Yes for particularly complex ones

FFT

Graphs

Screw order - just write it!

---

Did the polynomial identiy notes above very Sloppily!

---

## Dynamic Programming

(wrote principles on cheat sheet)

longest palidrone sub seq

memoize!

---

# of BST options explode fast

---

like AI chess

(18)

10/11
24

# Shortest Path

finally the graph stuff

## Single Source Shortest Path

table

Should quickly review

representations    adj list

adj matrix → 1 if connected

incidence matrix → shows if things are related

## BFS

Colors white = untard

$\pi$ = parent

enqueue things when see

when visit make grey

when done w/ children → black

19

## Shortest path distance

$\delta(s,v) \le \delta(s,u)+1$

if an edge $(u,v)$

So can prove w/ induction

Can create a predicessor subgraph

$\pi$ = parent

produces a tree

## DFS

predicessor subgraph can have several trees

forms a depth first forest

Same colors

- black when its adj list done

20

discovery + finishing time

* recurses down
(This code is not queue/stack but
recursive I think)

Tree edges normal

back edge about to make a loop

forward edge connecting a path we will
discover elsewhere

right, was alway confused here

Cross edge all other edges

Topo sort
Call DFS
sort by finishing time

21)

## Minimum Spanning Tree

What I actually need to study

Connect n pts w/ n-1 edges

or least amt of "string"

Sum of edge length

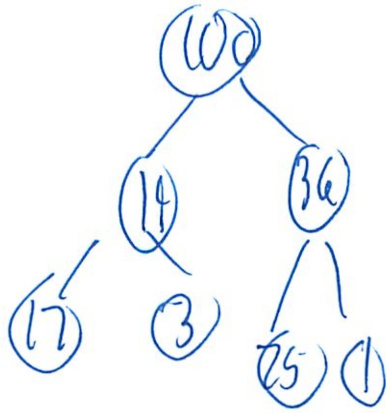
Or min weight/cost edges

Kruskal O(E lg V)

Prim " but O(E + V lg V) w/ fib heaps

### binary heaps

ah the normal binary tree



ah no - only top level matters!

22

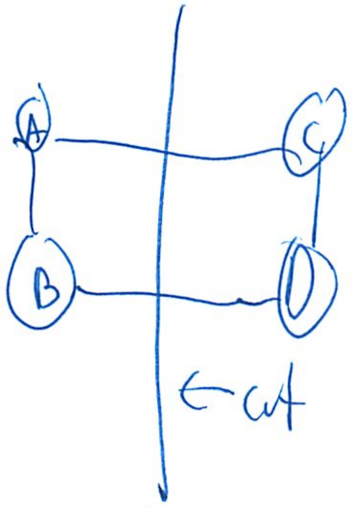
both are greedy *

↑locally optimal is globally optimal

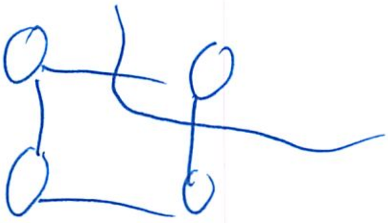
Prior to each iteration, A is a subset of some min spanning tree

find the edge we can add that does not violate

cut = partition of graph



or



Crosses cut if m on one side and n on other and edge (m,n)

cut respects ~~a~~ set of edge if all on one side

23

light edge if crossing the cut if has the min weight of an edge

- can be > 1

↳ Or more generally, the lowest cost edge in a set of edges

---

Kruskal use specific rule to pick safe edge

finds safe edge to add by finding of all edges connecting 2 trees ~~on~~ in the forest (u, v) of least weight

Uses disjoint-set data structure

call find-set(u) find set(v)

## disjoint Set data structure

set els partitioned into 2+ non overlapping subsets

aka union-find
merge-find

each set represented by linked list (naive)

or tree data structure
↑ can be a binary tree
are like digraphs
forest = collection of ~~[illegible]~~ trees
↑ but no common root (or would be a tree)

find follows parent node till reaches root
↳ ie find root?

Union combines trees by ~~merging~~ ~~[illegible]~~ attaching root of one to root of other this is no better than linked list

↳ but if we balance...

1. Union by rank always attach smaller tree to root of larger tree

↳ this only ↑ depth if depths =

and since tree running times based on depth

One el tree = rank = 0

rank ≈ depth

↳ except for path compression

2. Path Compression flaten structure when calling find

So as it goes up, sets each's parent

to the overall root of tree

⟶ so running time ~ O(1) when we do both

Psudo code on cheat sheet

What does it mean to be in same / diff trees?

A = forest being grown

* at start we have V trees

  L one for each vertex

So like bilding tree up wards

  Combining pieces

---

* Also note that edges ~~[illegible]~~ don't need to be adj to be added

  L for some reason

---

I wish they would show you the data structure

---

∴ Do they mean $A = A \cap \{u, v\}$ not $\cup$ ???

27

**Prim**

much like Dijstra

**Dijstra**

weighted all non neg

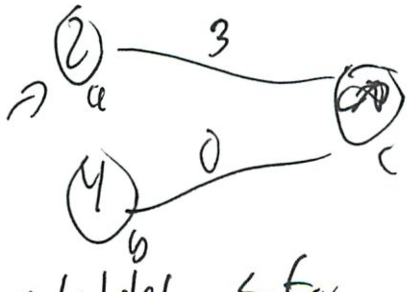
keep extracting min from priority queue

then relax (u, v, w)

greedy structure

**Relax edges** (u, v, w)

Can we improve shortest existing path to v by using the edge (u, v)



min cost we've calculated so far

28

So we do relax (a, c)

replace c with 5

relax (b, c)

replace c w/ 4

also has some parent thing it tracks

edges in set A always form single tree

└ opposite of what we saw earlier!

always connects to a new isolated vertex

need a way to find what to add

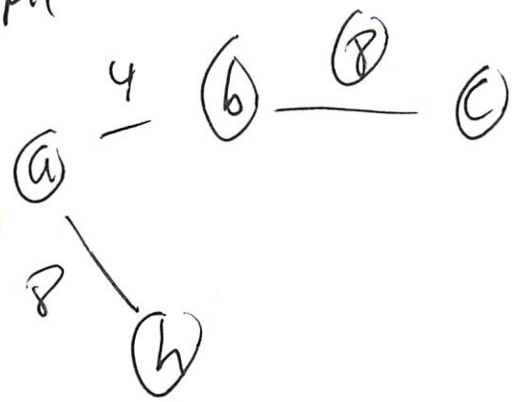
└ min priority queue

using key, = min weight of any edge connecting v to a vertex in the tree

29

So $A = \{(v, v.\pi) : v \in V - \{r\} - Q\}$

in example



Why ~~not~~

a

b

but then why not h, insead of c

; So min to any pt and they just picked randomly;

Plus need to prove it works

$O(E \lg V)$

but w/ Fib heaps $O(E + V \lg V)$

30

# Bellman Ford

Single source shortest path
edges can be neg

```
for every vertex
    for every edge
        relax
 if ⊖ loop → false
```

## All-Pairs Shortest Paths

Dij $O(V^3)$ we can do much better

matrix of edge weights

slow nieve method $O(V^4)$

Skip
was in lecture: what is the _last_ edge traversed on a u → v path

31

I figured this out earlier in the semester - can I remember

## Floyd Warshall $O(V^3)$

intermediate vertices

$$d_{ij}^{(k)} = \begin{cases} w_{ij} \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) \end{cases}$$

So k is a specific node I'm pretty sure

bottom up or top down

think I get it

was good I studied this earlier

unlike min spanning tree

---

## Matrix Multiplication $O(n^3 \lg n)$

$C = A \circ B$

$$C_{ij} = \sum_{k=1}^{n} A_{ik} + B_{kj}$$

$$D^{(k)} = D^{(k-1)} \odot W = \ldots = W^{(n)}$$

I don't get this...

---

Johnson

best alg

esp good on sparse

not DP

add a constant to avoid $\ominus$ weights

$O(V^2 \lg V + VE)$ ~~da~~stra

but no $\ominus$ weight cycles allowed

---

Computes using Bellman Ford + Dij algs

~~Ohh now I remember!~~

$$\hat{\delta}(u,v) \;\Big/\; \delta(u,v)$$

slash

33

running Dij once for each

---

Shipping rec 3

---

Network Flow

this is what I remembered

Graph

w/ capacity

and current flow

→ flow ≤ capacity

→ flow in = flow out at node

except source, sink

→ flow out of source = flow into sink

If self loop turn to dummy node

I don't know why ...

Cuts = incremental flow w/ augmented path

as long as source / sink are seperate

$$f(S,T) \leq \underset{?\text{cut}}{c(S,T)}$$

Residual network

$$c_{f}(u,v) = c(u,v) - f(u,v) > 0$$

Augmented path

any path s to t in $G_f$ is aug path

Can turn into reachability condition

if reach t augmenting path, can ↑ max flow

alg 2: Ford Fulkerson

↳ says won't be on quiz!

35

Find some augmenting path p and use it to modify the flow f

$|f| + |f_p|$

Then we update the flow for each edge

*Design and Analysis of Algorithms* March 4, 2011
Massachusetts Institute of Technology 6.046J/18.410J
Professors Charles E. Leiserson and Dana Moshkovitz Practice Quiz 1

# Practice Quiz 1

- Do not open this quiz booklet until you are directed to do so. Read all the instructions first.
- The quiz contains multi-part problems. You have 60 minutes to earn 60 points.
- This quiz booklet contains **8 double-sided** pages, including this one and a double-sided sheet of scratch paper; there should be 12 (numbered) pages of problems.
- This quiz is closed book. You may use one double sided Letter ($8\frac{1}{2}'' \times 11''$) or A4 crib sheet. No calculators or programmable devices are permitted. Cell phones must be put away.
- Write your solutions in the space provided. Extra scratch paper may be provided if you need more room, although your answer should fit in the given space.
- Do not waste time re-deriving facts that we have studied. It is sufficient to cite known results.
- Do not spend too much time on any one problem. Generally, a problem's point value is an indication of how much time to spend on it.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

| Problem | Points | Grade | Initials |
|---|---|---|---|
| 1 | 8 | | |
| 2 | 10 | | |
| 3 | 10 | | |
| 4 | 10 | | |
| 5 | 10 | | |
| 6 | 12 | | |
| Total | | | |

**Name:** ______________________________

Circle your recitation:

| | | | | | |
|---|---|---|---|---|---|
| R01 | R02 | R03 | R04 | R05 | R06 |
| F10 | F11 | F12 | F1 | F2 | F3 |
| Michael | Michael | Prasant | Prasant | Szymon | Szymon |

## Probability Toolkit

1. **Markov Inequality:** For any non-negative random variable $X$, $\Pr\{X \geq \lambda\} \leq \mathrm{E}[X]/\lambda$.

2. **Chernoff Bounds:** Let $X_1, X_2, X_3 \ldots X_n$ be independent Bernoulli trails such that, for $1 \leq i \leq n$, $\Pr\{X_i = 1\} = p_i$, where $0 < p_i < 1$. Then, for $X = \sum_{i=1}^{n}$, $\mu = \mathrm{E}[X] = \sum_{i=1}^{n} p_i$,

$$\Pr\{X < (1-\epsilon)\mu\} < \begin{cases} e^{-\mu\epsilon^2/2} & \text{for } 0 < \epsilon \leq 1 \\ 2^{-(1+\epsilon)\mu} & \text{for } \epsilon > 2e - 1 \end{cases}$$

**Problem 1. Recurrences** Solve the following recurrences by giving tight $\Theta$-notation bounds. As usual, assume that $T(n) = O(1)$ for $n \leq 2$. [8 points] (2 parts)

**(a)** [4 points] $T(n) = 9T(\sqrt[3]{n}) + \Theta(\log(n))$.

**(b)** [4 points] $T(n) = 7T(n/2) + \Theta(n)$.

**Problem 2. True or False, and Justify** [10 points] (5 parts)

Circle **T** or **F** for each of the following statements, and briefly explain why. Your justification is worth more points than your true-or-false designation.

**(a) T F** [2 points] Michael is working on his thesis and is stuck on the following problem: He has pairs of DNA sequences and a priority (the distribution of priorities is uniform over $[0, 1]$) associated with each DNA sequence. He would like to insert, delete pairs dynamically and search through $N$ DNA sequences in $O(\log N)$ expected time and at the same time retrieve the DNA sequence with the maximum priority in $O(1)$ time in the worst case. He hears about TREAPS from a student enrolled in 6.046 and claims he has completed his thesis. Is he right?

**(b) T F** [2 points] A static set of $n$ elements can be stored in a hash table that uses $O(n)$ space and supports look up in $O(1)$ time in the worst case.

**(c) T F** [2 points] A Monte Carlo algorithm always runs in deterministic time.

**(d) T F** [2 points] Suppose we have computed a minimum spanning tree (MST) and its total weight for some graph $G = (V, E)$. If we make a new graph $G'$ by adding 1 to the weight of every edge in $G$, we will need to spend $\Omega(E)$ time to compute an MST and its total weight for the new graph $G'$.

**(e)** **T F** [2 points] A data structure $D$ allows insertions, deletions and search in $O(1)$ amortized time. Imagine the state of $D$ after $n$ insertions and $m$, $m >> n$, searches. Then, $D$ cannot take $\Omega(n^2)$ time for any deletion.

**Problem 3. Boosting the probability** (2 parts) [10 points]

**(a)** [5 points] Consider a Randomized algorithm A which is always correct when it outputs YES while it may tag a YES instance as a NO with probability $1/3$. It has a runtime of $O(n \log n)$ for an input instance of size $n$. Can you amplify the probability of success to $1 - O(1/n^2)$? If yes, give an upper bound on the running time required to achieve it. Otherwise, give a justification as to why it is not possible to amplify probability of success.

**(b)** [5 points] $\Pi$ is a Randomized algorithm that has an error probability of $1/4$. That is, it is a two sided error algorithm and hence outputs both false positives and false negatives. Can you design a new algorithm $\Pi'$ which has the same functionality as $\Pi$ but an error probability of $2^{-c}$, where $c > 3$ is a constant? If so, how are the run times of $\Pi$ and $\Pi'$ related? Otherwise, give a justification as to why one cannot construct $\Pi'$.

**Problem 4. Binary Counting** (5 parts) [10 points] Consider the problem of implementing a $k$-bit binary counter that counts upward from 0. We use an array $A[0 \ldots k-1]$ of bits, where $A.length = k$, as the counter. The least significant bit is stored in $A[0]$. So, $x = \sum_{i=0}^{k-1} A[i].2^i$. Initially, $x = 0$, and thus $A[i] = 0$ for $i = 0, 1 \ldots n-1$. To add $1 (modulo 2^k)$ to the value in the counter, we use the following procedure.

```
INCREMENT(A):
$i = 0$
while $i <$ A.length and A[i] == 1
    A[i] =0
    $i = i + 1$
if $i <$ A.length
    A[i]=1
```

**(a)** [2 points] Argue that A[1] is flipped only every time the counter is incremented. Extend the above argument to show that the bit A[i] is flipped every $2^i$-th time for $i \geq 0$.

**(b)** [2 points] Use an aggregate analysis to conclude that the total work performed for $n$ INCREMENT operations is $O(n)$ in the worst case.

**(c)** [2 points] Show that if a DECREMENT operation were included in the $k$-bit counter, $n$ operations could cost as much as $\Theta(nk)$ time.

**(d)** [2 points] Use a potential function argument, with the number of $1s$ in the counter after the operation as the potential function, to prove that each INCREMENT operation cost $O(1)$ amortized time.

**(e)** [2 points] Suppose that a counter begins at a number with $b$ $1s$ in its binary representation, rather than at $0$. Show that the cost of performing $n$ INCREMENT operations is $O(n)$ if $n = \Omega(b)$. (Do not assume that $b$ is constant.)

**Problem 5.** (2 parts) [10 points] Suppose that you are given an array $A$ of $n$ bits that is either of type 1: contains half zeros and half ones in some arbitrary order or of type 2: contains $2n/3$ zeros and $n/3$ ones in some arbitrary order. You are given either a type 1 or a type 2 array with equal probability. Your goal is to determine whether $A$ is type 1 or type 2.

**(a)** [5 points] Give an exact lower bound in terms of $n$ (not using asymptotic notation) on the worst-case running time of any deterministic algorithm that solves this problem.

**(b)** [5 points] Consider the following randomized strategy: Choose uniformly at random an element from the given array. If the element is 0, it outputs "type 2" else it outputs "type 1". Show that this algorithm makes an error with probability at most $1/2$.

**Problem 6. Universal Hashing** (4 parts) [12 points]

Recall that a collection $\mathcal{H}$ of hash function from a universe $\mathcal{U}$ to a range $R$ is called **universal** if for all $x \neq y$ in $\mathcal{U}$ we have

$$\Pr_{h \in \mathcal{H}}[\, h(x) = h(y) \,] = \frac{1}{|R|}$$

We want to implement universal hashing from $\mathcal{U} = \{0,1\}^p$ to $R = \{0,1\}^q$ (where $p > q$).

For any $q \times p$ boolean matrix $A$ and any $q$-bit vector $b$ we define the function $h_{A,b} : \{0,1\}^p \to \{0,1\}^q$ as $h_{A,b}(x) = Ax + b$, where by this we mean the usual matrix-vector multiplication and the usual vector addition, except that all the operations are done modulo 2. For example, if $q = 2, p = 3$ and

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix},$$

we have

$$h_{A,b}(x) = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x_2 + x_3 + 1 & mod\ 2 \\ x_1 + x_3 & mod\ 2 \end{pmatrix}$$

Let us establish that the hash family $\mathcal{H}$ is indeed universal for the specified range.

Notice that for any function $h_{A,b}$ we have $h(x) = h(y)$ if and only if $A(x - y) = \bar{0}$. We want to show that for any non-zero vector $z (= x - y) \in \{0,1\}^p$, if we choose $A$ at random from $\{0,1\}^{q \times p}$ then the probability of getting $Az = \bar{0}$ is exactly $1/2^q$.

So let $z$ be any non-zero vector in $\{0,1\}^p$, and assume w.l.o.g. that the first coordinate of $z$ is non-zero (the same argument holds if we assume that any other coordinate of $z$ is non-zero). In other words, we assume that $z_1 = 1$.

Denote $A = \{a_{ij}\}$ and consider any choice of all the elements in $A$ except for the first column. That is, we assume that we have already chosen all the elements

$$\begin{matrix} a_{12} & a_{13} & \cdots & a_{1p} \\ & & \vdots & \\ a_{q2} & a_{q3} & \cdots & a_{qp} \end{matrix}$$

and the only free variables left are $a_{11}, a_{21}, \cdots a_{q1}$.

**(a)** [3 points] Argue that in order to satisfy $Az = \bar{0}$, these $a_{i1}$'s need to satisfy $a_{i1}z_1 + a_{i2}z_2 + \cdots + a_{ip}z_p = 0 \pmod{2}$ for all $i$.

**(b)** [3 points] Conclude that for $z_1 = 1$, the only choice which will satisfy $Az = \bar{0}$ is

$$\begin{aligned} a_{11} = &\quad -(a_{12}z_2 + \cdots + a_{1n}z_p) \pmod{2} \\ &\vdots \\ a_{q1} = &\quad -(a_{q2}z_2 + \cdots + a_{qn}z_p) \pmod{2} \end{aligned}$$

**(c)** [3 points] Show that the probability of hitting the only value satisfying $Az = \bar{0}$ is $1/2^q$ and conclude that $\mathcal{H}$ is an universal hash family from $\mathcal{U}$ to $\mathcal{R}$.

**(d)** [3 points] Let $S \subseteq \mathcal{U}$ be the set we would like to hash. Let $n = |S|$ and $m = 2^q$. Prove that if we choose $h_{A,b}$ from $\mathcal{H}$ uniformly at random, the expected number of pairs $(x, y) \in S \times S$ with $x \neq y$ and $h_{A,b}(x) = h_{A,b}(y)$ is $O(\frac{n^2}{m})$.

SCRATCH PAPER

SCRATCH PAPER

*Design and Analysis of Algorithms* March 4, 2011
Massachusetts Institute of Technology 6.046J/18.410J
Professors Charles E. Leiserson and Dana Moshkovitz Practice Quiz 1

# Practice Quiz 1

- Do not open this quiz booklet until you are directed to do so. Read all the instructions first.
- The quiz contains multi-part problems. You have 60 minutes to earn 60 points.
- This quiz booklet contains **8 double-sided** pages, including this one and a double-sided sheet of scratch paper; there should be 12 (numbered) pages of problems.
- This quiz is closed book. You may use one double sided Letter ($8\frac{1}{2}'' \times 11''$) or A4 crib sheet. No calculators or programmable devices are permitted. Cell phones must be put away.
- Write your solutions in the space provided. Extra scratch paper may be provided if you need more room, although your answer should fit in the given space.
- Do not waste time re-deriving facts that we have studied. It is sufficient to cite known results.
- Do not spend too much time on any one problem. Generally, a problem's point value is an indication of how much time to spend on it.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

| Problem | Points | Grade | Initials |
|---|---|---|---|
| 1 | 8 | | |
| 2 | 10 | | |
| 3 | 10 | | |
| 4 | 10 | | |
| 5 | 10 | | |
| 6 | 12 | | |
| Total | | | |

**Name:** ______________________________

Circle your recitation:

| | | | | | |
|---|---|---|---|---|---|
| R01 | R02 | R03 | R04 | R05 | R06 |
| F10 | F11 | F12 | F1 | F2 | F3 |
| Michael | Michael | Prasant | Prasant | Szymon | Szymon |

## Probability Toolkit

1. **Markov Inequality:** For any non-negative random variable $X$, $\Pr\{X \geq \lambda\} \leq \mathrm{E}[X]/\lambda$.

2. **Chernoff Bounds:** Let $X_1, X_2, X_3 \ldots X_n$ be independent Bernoulli trails such that, for $1 \leq i \leq n$, $\Pr\{X_i = 1\} = p_i$, where $0 < p_i < 1$. Then, for $X = \sum_{i=1}^{n}$, $\mu = \mathrm{E}[X] = \sum_{i=1}^{n} p_i$,

$$\Pr\{X < (1-\epsilon)\mu\} < \begin{cases} e^{-\mu\epsilon^2/2} & \text{for } 0 < \epsilon \leq 1 \\ 2^{-(1+\epsilon)\mu} & \text{for } \epsilon > 2e-1 \end{cases}$$

**Problem 1. Recurrences** Solve the following recurrences by giving tight $\Theta$-notation bounds. As usual, assume that $T(n) = O(1)$ for $n \leq 2$. [8 points] (2 parts)

(a) [4 points] $T(n) = 9T(\sqrt[3]{n}) + \Theta(\log(n))$.

**Solution:** Let $n = 2^m$. Then the recurrence becomes $T(2^m) = 9T(2^{m/3}) + \Theta(m)$. Setting $S(m) = T(2^m)$ gives us $S(m) = 9S(m/3) + \Theta(m)$. Using case 1 of the Master's Method gives us $S(m) = \Theta(m^2)$ or $T(n) = \Theta(\log^2 n)$

(b) [4 points] $T(n) = 7T(n/2) + \Theta(n)$.

**Solution:** Master's theorem. $T(n) = \Theta(n^{\lg 7})$.

**Problem 2. True or False, and Justify** [10 points] (5 parts)

Circle **T** or **F** for each of the following statements, and briefly explain why. Your justification is worth more points than your true-or-false designation.

**(a) T F** [2 points] Michael is working on his thesis and is stuck on the following problem: He has pairs of DNA sequences and a priority (the distribution of priorities is uniform over $[0, 1]$) associated with each DNA sequence. He would like to insert, delete pairs dynamically and search through $N$ DNA sequences in $O(\log N)$ expected time and at the same time retrieve the DNA sequence with the maximum priority in $O(1)$ time in the worst case. He hears about TREAPS from a student enrolled in 6.046 and claims he has completed his thesis. Is he right?

**Solution:** True. Treaps serve the same purpose. Since the distribution of priorities is uniform, the expected tree height is $O(\log N)$ which gives us the desired expected time for search.

**(b) T F** [2 points] A static set of $n$ elements can be stored in a hash table that uses $O(n)$ space and supports look up in $O(1)$ time in the worst case.

**Solution:** True. Perfect Hashing.

**(c) T F** [2 points] A Monte Carlo algorithm always runs in deterministic time.

**Solution:** True. A Monte Carlo algorithm always runs in deterministic time. Its output, however, may not be always correct.

**(d) T F** [2 points] Suppose we have computed a minimum spanning tree (MST) and its total weight for some graph $G = (V, E)$. If we make a new graph $G'$ by adding 1 to the weight of every edge in $G$, we will need to spend $\Omega(E)$ time to compute an MST and its total weight for the new graph $G'$.

**Solution:** False. If $T$ is an MST for $G$ with weight $w$, then it is also an MST for $G'$ with weight $w + |V| - 1$.

**(e) T F** [2 points] A data structure $D$ allows insertions, deletions and search in $O(1)$ amortized time. Imagine the state of $D$ after $n$ insertions and $m$, $m >> n$, searches. Then, $D$ cannot take $\Omega(n^2)$ time for any deletion.

**Solution:** False. An amortized bound does not guarantee worst case time bounds on the execution of any single operation. Here, the amortized bound only guarantees that the next deletion cannot take more than $O(n + m)$ time, which is the worst-case bound for the whole sequence.

**Problem 3. Boosting the probability** (2 parts) [10 points]

**(a)** [5 points] Consider a Randomized algorithm A which is always correct when it outputs YES while it may tag a YES instance as a NO with probability $1/3$. It has a runtime of $O(n \log n)$ for an input instance of size $n$. Can you amplify the probability of success to $1 - O(1/n^2)$? If yes, give an upper bound on the running time required to achieve it. Otherwise, give a justification as to why it is not possible to amplify probability of success.

**Solution:** Yes. The answer follows the construction given in pset problem 3-1(d). Let A' be an algorithm which runs algorithm A $k$ independent times. A' returns YES if any instance of A returns YES and returns NO otherwise. Thus, A' is always correct when it returns YES while it may tag a YES instance as a NO with probability $(1/3)^k$. The probability of success of A' is $1 - (1/3)^k$. If $k = 2\log_3 n$, then the probability of success is $1 - 1/n^2 = 1 - O(1/n^2)$. A' runs in $O(n \log^2 n)$ time.

**(b)** [5 points] $\Pi$ is a Randomized algorithm that has an error probability of $1/4$. That is, it is a two sided error algorithm and hence outputs both false positives and false negatives. Can you design a new algorithm $\Pi'$ which has the same functionality as $\Pi$ but an error probability of $2^{-c}$, where $c > 3$ is a constant? If so, how are the run times of $\Pi$ and $\Pi'$ related? Otherwise, give a justification as to why one cannot construct $\Pi'$.

**Solution:** It is possible to construct $\Pi'$. To achieve the necessary bounds, we run $\Pi$ $t$ times and then take a majority, where $t$ is a parameter that comes out from the analysis.

If we execute $\Pi$ $t$ times, the expected number of correct answers is $3t/4$. However, we cannot be sure about the deviations from the expectation – which can be huge – so we need something more stronger. Specifically, we need to show that the deviation from the mean is also small. In particular, we will need to show that the probability we see less than $t/2$ successes is small (in our case $2^{-c}$).

Let $X$ denote the number of successes. Thus, we would like to bound the $\Pr\{X < t/2\}$. Since, each of the $t$ executions are independent Bernoulli random variables we can use Chernoff bounds over the range $\epsilon \in (0, 1)$.

$$\Pr\{X < (1-\epsilon)\mu\} < e^{-\mu\epsilon^2/2}$$
$$\Pr\{X < (1-\epsilon)3t/4\} < e^{-3t\epsilon^2/8}$$

Set $\epsilon = 1/3$ to bound

$$\Pr\{X < t/2\} < e^{-t/24}$$

We request that

$$e^{-t/24} < 2^{-c}$$

which is true when

$$t > 24c \ln 2$$

Thus, algorithm $\Pi'$ executes $\Pi$ $24c \ln 2$ times and then takes the majority answer. The resulting algorithm has an error probability smaller than $2^{-c}$ and running time $24c \ln 2$ times longer than the original algorithm.

*Remark on Chernoff Bounds:* It is important to distinguish the cases when you need Chernoff and the cases which do not need Chernoff like part (a). Chernoff bounds are extremely useful if you have independent Bernoulli random variables and your interest is to bound the deviation from expectation.

**Problem 4. Binary Counting** (5 parts) [10 points] Consider the problem of implementing a $k$-bit binary counter that counts upward from 0. We use an array $A[0 \ldots k-1]$ of bits, where $A.length = k$, as the counter. The least significant bit is stored in $A[0]$. So, $x = \sum_{i=0}^{k-1} A[i].2^i$. Initially, $x = 0$, and thus $A[i] = 0$ for $i = 0, 1 \ldots n-1$. To add $1(modulo 2^k)$ to the value in the counter, we use the following procedure.

```
INCREMENT(A):
$i = 0$
while $i <$ A.length and A[i] == 1
    A[i] = 0
    $i = i + 1$
if $i <$ A.length
    A[i]=1
```

**(a)** [2 points] Argue that A[1] is flipped only every time the counter is incremented. Extend the above argument to show that the bit A[i] is flipped every $2^i$-th time for $i \geq 0$.

**Solution:** It is easy to see that the first bit A[0] is flipped every time the counter is incremented. A[1] is flipped only when A[0]=1, which is every other increment operation. In general for $i \geq 1$, A[i] is flipped only when A[j]=1 for all $0 \leq j < i$. This only happens once out of every $2^i$ possibilities for $A[0], \ldots, A[i-1]$.

**(b)** [2 points] Use an aggregate analysis to conclude that the total work performed for $n$ INCREMENT operations is $O(n)$ in the worst case.

**Solution:** The total cost is equal to the number of times a bit is flipped. From part (a) we know that A[i] will flip a total of $\lfloor n/2^i \rfloor$.

$$\begin{aligned}
\sum_0^n c_i &= \sum_0^k \text{Number of times A[i] is flipped} \\
&= \sum_0^k \lfloor n/2^i \rfloor \\
&\le \sum_0^k n/2^i \\
&= n \sum_0^k 1/2^i \\
&\le 2n \\
&= O(n)
\end{aligned}$$

**(c)** [2 points] Show that if a DECREMENT operation were included in the $k$-bit counter, $n$ operations could cost as much as $\Theta(nk)$ time.

**Solution:** Consider a sequence of operation that begins with DECREMENT and then alternates between INCREMENT and DECREMENT. Then, the counter alternates between all 0's and all 1's. Each operation changes every bit and costs $\Theta(k)$ time. The total cost is $\Theta(nk)$.

**(d)** [2 points] Use a potential function argument, with the number of 1s in the counter after the operation as the potential function, to prove that each INCREMENT operation cost $O(1)$ amortized time.

**Solution:** Let $A_i$ be the state of the counter after the $i$-th operation, $\Phi(A_i)$ be the number of 1's in $A_i$, and $\hat{c}_i$ be the amortized cost of the $i$-th operation. The total amortized cost is

$$\sum_0^n \hat{c}_i = \sum_0^n c_i + \Phi(A_n) - \Phi(A_0)$$

Since $\Phi(A_0) = 0$ and $\Phi(A_i) \ge 0$, the total amortized cost is an upper bound on the actual total cost. If $A_{i-1}$ is filled with all 1's, then $\Phi(A_i) = 0$, $\Phi(A_{i-1}) = k$, and $c_i = k$. In this case, $\hat{c}_i = 0$. Otherwise if $A_{i-1}$ is not filled with 1's, then operation $i$ will flip one bit from 0 to 1 and flip $c_i - 1$ bits from 1 to 0. In this case, $\Phi(A_i) - \Phi(A_{i-1}) = c_i - 2$ and $\hat{c}_i = 2$. The amortized cost is always $O(1)$.

**(e)** [2 points] Suppose that a counter begins at a number with $b$ 1$s$ in its binary representation, rather than at 0. Show that the cost of performing $n$ INCREMENT operations is $O(n)$ if $n = \Omega(b)$. (Do not assume that $b$ is constant.)

**Solution:** If we use the same potential function, then $\Phi(A_0) = b$, and the potential difference $\Phi(A_i) - \Phi(A_0) \ge -b$ may now be negative. However even though $\sum \hat{c}_i$ may no longer be an upper bound on the total actual cost, we have a new upper bound:

$$\sum c_i \le \sum \hat{c}_i + \Phi(A_0) = O(n + b)$$

If $n = \Omega(b)$, then the total cost is still $O(n)$.

**Problem 5.** (2 parts) [10 points] Suppose that you are given an array $A$ of $n$ bits that is either of type 1: contains half zeros and half ones in some arbitrary order or of type 2: contains $2n/3$ zeros and $n/3$ ones in some arbitrary order. You are given either a type 1 or a type 2 array with equal probability. Your goal is to determine whether $A$ is type 1 or type 2.

**(a)** [5 points] Give an exact lower bound in terms of $n$ (not using asymptotic notation) on the worst-case running time of any deterministic algorithm that solves this problem.

**Solution:** Any correct deterministic algorithm must look at exactly $5n/6 + 1$ entries in the worst case. Otherwise, the adversary can show it $n/3$ ones, and $n/2$ zeros ($5/6n$ entries). The remaining $n/6$ elements will either be all ones (type 1) or all zeros (type 2) and the algorithm must make another test.

**(b)** [5 points] Consider the following randomized strategy: Choose uniformly at random an element from the given array. If the element is 0, it outputs "type 2" else it outputs "type 1". Show that this algorithm makes an error with probability at most 1/2.

**Solution:** The algorithm makes an error if it picks the wrong type.

$$\begin{aligned}\Pr(\text{error}) &= \Pr(\text{output type} \neq \text{actual type}) = \\ &= \Pr(\text{drawn } 0 \wedge \text{type } 1) + \Pr(\text{drawn } 1 \wedge \text{type } 2) = \\ &= \Pr(\text{drawn } 0|\text{type } 1)\Pr(\text{type } 1) + \Pr(\text{drawn } 1|\text{type } 2)\Pr(\text{type } 2) = \\ &= \frac{1}{2}\cdot\frac{1}{2} + \frac{1}{3}\cdot\frac{1}{2} = \frac{5}{12}\end{aligned}$$

**Problem 6. Universal Hashing** (4 parts) [12 points]

Recall that a collection $\mathcal{H}$ of hash function from a universe $\mathcal{U}$ to a range $R$ is called **universal** if for all $x \neq y$ in $\mathcal{U}$ we have

$$\Pr_{h \in \mathcal{H}}[\, h(x) = h(y)\,] = \frac{1}{|R|}$$

We want to implement universal hashing from $\mathcal{U} = \{0,1\}^p$ to $R = \{0,1\}^q$ (where $p > q$).

For any $q \times p$ boolean matrix $A$ and any $q$-bit vector $b$ we define the function $h_{A,b} : \{0,1\}^p \rightarrow \{0,1\}^q$ as $h_{A,b}(x) = Ax + b$, where by this we mean the usual matrix-vector multiplication and the usual vector addition, except that all the operations are done modulo 2. For example, if $q = 2, p = 3$ and

$$A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix},$$

we have

$$h_{A,b}(x) = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x_2 + x_3 + 1 & mod\ 2 \\ x_1 + x_3 & mod\ 2 \end{pmatrix}$$

Let us establish that the hash family $\mathcal{H}$ is indeed universal for the specified range.

Notice that for any function $h_{A,b}$ we have $h(x) = h(y)$ if and only if $A(x - y) = \bar{0}$. We want to show that for any non-zero vector $z (= x - y) \in \{0,1\}^p$, if we choose $A$ at random from $\{0,1\}^{q \times p}$ then the probability of getting $Az = \bar{0}$ is exactly $1/2^q$.

So let $z$ be any non-zero vector in $\{0,1\}^p$, and assume w.l.o.g. that the first coordinate of $z$ is non-zero (the same argument holds if we assume that any other coordinate of $z$ is non-zero). In other words, we assume that $z_1 = 1$.

Denote $A = \{a_{ij}\}$ and consider any choice of all the elements in $A$ except for the first column. That is, we assume that we have already chosen all the elements

$$\begin{matrix} a_{12} & a_{13} & \cdots & a_{1p} \\ & & \vdots & \\ a_{q2} & a_{q3} & \cdots & a_{qp} \end{matrix}$$

and the only free variables left are $a_{11}, a_{21}, \cdots a_{q1}$.

**(a)** [3 points] Argue that in order to satisfy $Az = \bar{0}$, these $a_{i1}$'s need to satisfy $a_{i1}z_1 + a_{i2}z_2 + \cdots + a_{ip}z_p = 0 \pmod 2$ for all $i$.

**Solution:** Follows from matrix multiplication.

**(b)** [3 points] Conclude that for $z_1 = 1$, the only choice which will satisfy $Az = \bar{0}$ is

$$\begin{aligned} a_{11} &= -(a_{12}z_2 + \cdots + a_{1n}z_p) \pmod 2 \\ &\vdots \\ a_{q1} &= -(a_{q2}z_2 + \cdots + a_{qn}z_p) \pmod 2 \end{aligned}$$

**Solution:** Substitute $z_1 = 1$ and solve for $a_{i1}$, for $1 \le i \le q$, from the equations in part (a).

**(c)** [3 points] Show that the probability of hitting the only value satisfying $Az = \bar{0}$ is $1/2^q$ and conclude that $\mathcal{H}$ is an universal hash family from $\mathcal{U}$ to $\mathcal{R}$.

**Solution:** There are $2^q$ possibilities for $a_{11}, \ldots, a_{q1}$ that are chosen uniformly at random. Only the choice described in part (b) satisfies $Az = \bar{0}$, and it is chosen with probability $1/2^q = 1/|R|$. It follows from the definition that $\mathcal{H}$ is a universal hash family.

**(d)** [3 points] Let $S \subseteq \mathcal{U}$ be the set we would like to hash. Let $n = |S|$ and $m = 2^q$. Prove that if we choose $h_{A,b}$ from $\mathcal{H}$ uniformly at random, the expected number of pairs $(x, y) \in S \times S$ with $x \neq y$ and $h_{A,b}(x) = h_{A,b}(y)$ is $O(\frac{n^2}{m})$.

**Solution:** For any 2 distinct elements $x, y \in S$, let $I_{xy}$ be the indicator variable for the event that $h_{A,b}(x) = h_{A,b}(y)$. Since $\mathcal{H}$ is universal, $E[I_{xy}] = \Pr\{h_{A,b}(x) = h_{A,b}(y)\} = \frac{1}{m}$. The expected number of pairs $(x, y) \in S \times S$ with $x \neq y$ and $h_{A,b}(x) = h_{A,b}(y)$ is therefore:

$$E[\sum_{x \neq y} I_{xy}] = \sum_{x \neq y} E[I_{xy}] = \frac{n(n-1)}{m} = O(\frac{n^2}{m}).$$

SCRATCH PAPER

SCRATCH PAPER

# Quiz 1

- Do not open this quiz booklet until you are directed to do so. Read all the instructions first.
- The quiz contains 7 problems, several with multiple parts. You have 80 minutes to earn 80 points.
- This quiz booklet contains 9 pages, including this one, and a sheet of scratch paper which can be detached.
- This quiz is closed book. You may use one double sided Letter ($8\frac{1}{2}'' \times 11''$) or A4 crib sheet. No calculators or programmable devices are permitted. Cell phones must be put away.
- On page 2 there are several useful inequalities. Please review them before you start working on the quiz.
- Write your solutions in the space provided. If you run out of space, continue your answer on the back of the same sheet and make a notation on the front of the sheet.
- Do not waste time deriving facts that we have studied. It is sufficient to cite known results.
- Do not spend too much time on any one problem. Generally, a problem's point value is an indication of how many minutes to spend on it.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Please be neat.
- Good luck!

| Problem | Title | Points | Parts | Grade | Initials |
|---|---|---|---|---|---|
| 0 | Name | 1 | 9 | | |
| 1 | True or False | 21 | 7 | | |
| 2 | S3L3CT | 12 | 1 | | |
| 3 | SWAT Team | 12 | 1 | | |
| 4 | FIFO $= 2 \times$ LIFO | 12 | 1 | | |
| 5 | Big Edges | 10 | 1 | | |
| 6 | Minimum Madness | 12 | 4 | | |
| Total | | 80 | | | |

**Name:** ______________________________

**Problem 0. Name.** [1 point] Write your name on every page of this exam booklet! Don't forget the cover.

## Possibly useful facts for elsewhere in the quiz

1. **Markov Inequality:** For any nonnegative random variable $X$, we have

$$\Pr\{X \geq \lambda\} \leq \mathrm{E}[X]/\lambda\,.$$

2. **Chernoff Bounds:** Let $X_1, \ldots, X_n$ be $n$ independent Boolean random variables. Suppose that for $i = 1, 2, \ldots, n$, we have $\Pr\{X_i = 1\} = \delta_i$ for $0 \leq \delta_i \leq 1$. Let $X = \sum_{i=1}^{n} X_i$ and $\delta = (1/n) \cdot \sum_{i=1}^{n} \delta_i$. Then, for any $\delta \leq \gamma \leq 1$,

$$\Pr\{X \geq \gamma n\} \leq e^{-2(\gamma-\delta)^2 n}.$$

Note that this is a generalization of the Chernoff bound we saw in class to the case of not necessarily identically distributed random variables.

3. **Harmonic series:**

$$\sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1)\,.$$

**Problem 1. True or False.** [21 points] (7 parts)

Circle **T** or **F** for each of the following statements to indicate whether the statement is true or false, respectively. You need not justify your answers, but since wrong answers will be penalized, do not guess unless you are reasonably sure.

**(a) T F** The recurrence $T(n) = 2T(\sqrt{n}) + \lg n$ has solution $T(n) = \Theta(\lg^2 n)$.

**(b) T F** The numbers $1, 2, \ldots, 10$ can be placed into a tree data structure such that the tree satisfies both the min-heap property and the binary-search-tree property at the same time.

**(c) T F** The following collection $\mathcal{H} = \{h_1, h_2, h_3\}$ of hash functions is universal, where each hash function maps the universe $U = \{A, B, C, D\}$ of keys into the range $\{0, 1, 2\}$ according to the following table:

| $x$ | $h_1(x)$ | $h_2(x)$ | $h_3(x)$ |
|---|---|---|---|
| $A$ | 1 | 0 | 2 |
| $B$ | 0 | 1 | 2 |
| $C$ | 0 | 0 | 0 |
| $D$ | 1 | 1 | 0 |

**(d) T F** Consider a sequence of $n$ INSERT operations, $2n$ DECREASE-KEY operations, and $\sqrt{n}$ EXTRACT-MIN operations on an initially empty Fibonacci heap. The total running time of all these operations is $\Theta(n \lg n)$ in the worst case.

**(e) T F** In the analysis of the disjoint-set data structure presented during lecture, once a node other than the root or a child of the root is block-charged, it will never again be path-charged.

**(f) T F** A van-Emde-Boas data structure can support FIND-MIN, FIND-MAX, SUCCESSOR, and PREDECESSOR operations over the set $\{1, 2, \ldots, 2^{\sqrt{n}}\}$ in $O(\log n)$ time.

**(g) T F** Let $G = (V, E)$ be a connected undirected graph with edge-weight function $w : E \rightarrow \{1, 2, \ldots, 10\,|E|\}$. Then a minimum spanning tree of $G$ can be constructed in $O(E \lg \lg V)$ time.

**Problem 2. S3L3CT.** [12 points]

Professor John Von Meanmann is implementing an algorithm to find the $i$th smallest of a set $S$ of $n$ distinct elements. The professor improvises on the traditional worst-case linear-time algorithm and organizes elements into groups of 3 instead of groups of 5, resulting in the following algorithm, called S3L3CT:

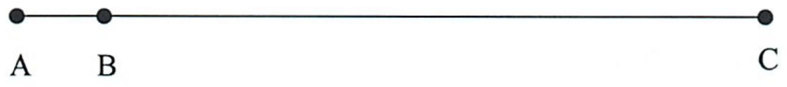
1. If $n = 1$, return the only element in the array.
2. Divide the $n$ elements of the input array into $\lfloor n/3 \rfloor$ groups of 3 elements each, with 0 to 2 elements left over.
3. Find the median of each of the $\lceil n/3 \rceil$ groups by rote.
4. Use S3L3CT recursively to find the median $x$ of the $\lceil n/3 \rceil$ medians found in Step 3.
5. Partition the input array around $x$. Let $k = \text{RANK}(x)$.
6. If $i = k$, then return $x$. Otherwise, use S3L3CT recursively to find the $i$th smallest element on the low side if $i < k$, or the $(i - k)$th smallest element on the high side if $i > k$.

State the recurrence for the running time $T(n)$ of S3L3CT running on an input of $n$ elements, and provide a tight asymptotic upper bound on its solution in terms of $n$. In order to simplify the math, assume that any set on which S3L3CT operates contains a multiple of 3 elements.

**Problem 3. SWAT Team.** [12 points]

Two swatsmen with fly swatters are located at arbitrary positions along a long corridor with many leaky windows. One at a time, houseflies appear at various locations along the corridor, and a swatsman goes to the location of the fly and swats it dead. The cost of a given strategy is the total distance traveled by the swatsmen. Argue that the greedy strategy of the closest swatsman going to the location of the fly is not $\alpha$-competitive for any finite $\alpha$.

(*Hint:* Consider the case that flies only appear at three locations $A$, $B$, and $C$, where $B$ falls between $A$ and $C$ and the distance from $A$ to $B$ is much smaller than the distance from $B$ to $C$, as shown below:



Consider the sequence $\langle C, A, B, A, B, A, B, A, B, A, \ldots \rangle$ of fly arrivals.)

**Problem 4. FIFO $= 2 \times$ LIFO.** [12 points]

A FIFO queue $Q$ supporting the operations ENQUEUE and DEQUEUE can be implemented using two stacks $S_1$ and $S_2$, each of which supports the operations PUSH, POP, and a test whether the stack is empty.

```
ENQUEUE($Q, x$):
    PUSH($S_1, x$)

DEQUEUE($Q$):
    if $S_1 = \emptyset$ and $S_2 = \emptyset$
        error "queue underflow"
    if $S_2 = \emptyset$
        while $S_1 \neq \emptyset$
            $x =$ POP($S_1$)
            PUSH($S_2, x$)
    return POP($S_2$)
```

Define a potential function $\Phi(Q) = c\,|S_1|$ for an appropriate constant $c > 0$, where $|S_1|$ is the number of items in $S_1$. Argue using a potential-function argument that each ENQUEUE and DEQUEUE operation takes $O(1)$ amortized time.

**Problem 5. Big Edges.** [10 points]

Let $G = (V, E)$ be a connected undirected graph with distinct edge weights $w : E \to \mathbb{R}$, and let $c$ be a cycle in $G$. Consider the edge $e$ on $c$ with the largest weight, that is, $w(e) \geq w(e')$ for all $e' \in c$. Prove that $e$ does not belong to the minimum spanning tree of $G$.

**Problem 6. Minimum Madness.** [12 points] (4 parts)

Consider the following program to find the minimum value in an array $A$ of $n$ distinct elements.

```
MINIMUM(A, n):
1  $min = \infty$ // Set $min$ to be a large value.
2  for $i = 1$ to $n$
3      if $min > A[i]$
4          $min = A[i]$
```

Assume that $A$'s elements are randomly permuted before invoking MINIMUM$(A, n)$ and that all permutations are equally likely. Let $X_i$ be the indicator random variable associated with the event that the variable $min$ is changed in line 4 during the $i$th iteration of the **for** loop, and let $Y = \sum_{i=1}^{n} X_i$ be the random variable denoting the total number of times $min$ is so updated.

**(a)** Argue that the probability that $A[i]$ is smaller than all the elements in $A[1 \,..\, i-1]$ is $1/i$.

**(b)** Show that $\mathrm{E}[X_i] = 1/i$.

**(c)** Prove that $\mathrm{E}[Y] = \Theta(\lg n)$.

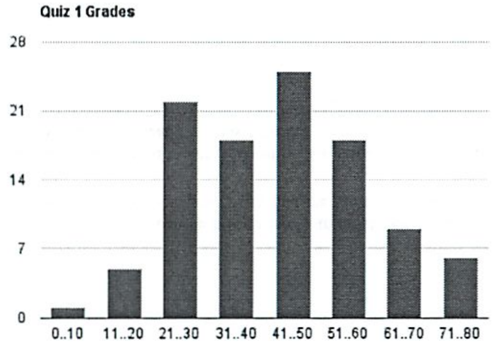
**(d)** Prove that for sufficiently large $n$, it holds that $\Pr\{Y \geq 5\mathrm{E}[Y]\} \leq 1/n^4$.

SCRATCH PAPER

*Design and Analysis of Algorithms*
Massachusetts Institute of Technology
Professors Charles E. Leiserson and Dana Moshkovitz

March 10, 2011
6.046J/18.410J
Quiz 1

# Quiz 1



**Problem 0. Name.** [1 point] Write your name on every page of this exam booklet! Don't forget the cover.

## Possibly useful facts for elsewhere in the quiz

1. **Markov Inequality:** For any nonnegative random variable $X$, we have

$$\Pr\{X \geq \lambda\} \leq \mathrm{E}[X]/\lambda .$$

2. **Chernoff Bounds:** Let $X_1, \ldots, X_n$ be $n$ independent Boolean random variables. Suppose that for $i = 1, 2, \ldots, n$, we have $\Pr\{X_i = 1\} = \delta_i$ for $0 \leq \delta_i \leq 1$. Let $X = \sum_{i=1}^{n} X_i$ and $\delta = (1/n) \cdot \sum_{i=1}^{n} \delta_i$. Then, for any $\delta \leq \gamma \leq 1$,

$$\Pr\{X \geq \gamma n\} \leq e^{-2(\gamma-\delta)^2 n} .$$

Note that this is a generalization of the Chernoff bound we saw in class to the case of not necessarily identically distributed random variables.

3. **Harmonic series:**

$$\sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1) .$$

**Problem 1. True or False.** [21 points] (7 parts)

Circle **T** or **F** for each of the following statements to indicate whether the statement is true or false, respectively. You need not justify your answers, but since wrong answers will be penalized, do not guess unless you are reasonably sure.

**(a) T F** The recurrence $T(n) = 2T(\sqrt{n}) + \lg n$ has solution $T(n) = \Theta(\lg^2 n)$.

**Solution:** False: Substitute $m = \lg n$. Then, $T(m) = 2T(m/2) + m = \Theta(m \lg m)$. So, $T(n) = \Theta(\lg n \lg \lg n)$ .

**(b) T F** The numbers $1, 2, \ldots, 10$ can be placed into a tree data structure such that the tree satisfies both the min-heap property and the binary-search-tree property at the same time.

**Solution:** True: Consider a tree that has only its rightmost branch, containing all the elements in monotonically increasing order .

**(c) T F** The following collection $\mathcal{H} = \{h_1, h_2, h_3\}$ of hash functions is universal, where each hash function maps the universe $U = \{A, B, C, D\}$ of keys into the range $\{0, 1, 2\}$ according to the following table:

| $x$ | $h_1(x)$ | $h_2(x)$ | $h_3(x)$ |
|---|---|---|---|
| $A$ | 1 | 0 | 2 |
| $B$ | 0 | 1 | 2 |
| $C$ | 0 | 0 | 0 |
| $D$ | 1 | 1 | 0 |

**Solution:** True: By verifying that any two rows do no have more than one element in common.

**(d) T F** Consider a sequence of $n$ INSERT operations, $2n$ DECREASE-KEY operations, and $\sqrt{n}$ EXTRACT-MIN operations on an initially empty Fibonacci heap. The total running time of all these operations is $\Theta(n \lg n)$ in the worst case.

**Solution:** False: The time by this sequence of operations is $O(n + 2n + \sqrt{n} \lg n) = O(n)$.

**(e) T F** In the analysis of the disjoint-set data structure presented during lecture, once a node other than the root or a child of the root is block-charged, it will never again be path-charged.

**Solution:** True: Follows from the definitions of block-charges and path-charges

.

**(f) T F** A van-Emde-Boas data structure can support FIND-MIN, FIND-MAX, SUCCESSOR, and PREDECESSOR operations over the set $\{1, 2, \ldots, 2^{\sqrt{n}}\}$ in $O(\log n)$ time.

**Solution:** True: vEB trees can perform the above operations in $O(\lg \lg (2^{\sqrt{n}})$ or $O(\log n)$ time.

**(g) T F** Let $G = (V, E)$ be a connected undirected graph with edge-weight function $w : E \to \{1, 2, \ldots, 10\,|E|\}$. Then a minimum spanning tree of $G$ can be constructed in $O(E \lg \lg V)$ time.

**Solution:** True: Use vEB data structure as heap.

**Problem 2. S3L3CT.** [12 points]

Professor John Von Meanmann is implementing an algorithm to find the $i$th smallest of a set $S$ of $n$ distinct elements. The professor improvises on the traditional worst-case linear-time algorithm and organizes elements into groups of 3 instead of groups of 5, resulting in the following algorithm, called S3L3CT:

1. If $n = 1$, return the only element in the array.
2. Divide the $n$ elements of the input array into $\lfloor n/3 \rfloor$ groups of 3 elements each, with 0 to 2 elements left over.
3. Find the median of each of the $\lceil n/3 \rceil$ groups by rote.
4. Use S3L3CT recursively to find the median $x$ of the $\lceil n/3 \rceil$ medians found in Step 3.
5. Partition the input array around $x$. Let $k = \text{RANK}(x)$.
6. If $i = k$, then return $x$. Otherwise, use S3L3CT recursively to find the $i$th smallest element on the low side if $i < k$, or the $(i - k)$th smallest element on the high side if $i > k$.

State the recurrence for the running time $T(n)$ of S3L3CT running on an input of $n$ elements, and provide a tight asymptotic upper bound on its solution in terms of $n$. In order to simplify the math, assume that any set on which S3L3CT operates contains a multiple of 3 elements.

**Solution:** $x$ is at least as large as $n/3$ elements (for half of the triplets, 2 out of 3 elements) and at least as small as $n/3$ elements (for the other half of the triplets, 2 out of 3 elements). So the recursion in Step 5 is, in the worst case, on an array of size $2n/3$. The recursion in Step 3 is always on $n/3$ elements. So we get the recurrence $T(n) = T(2n/3) + T(n/3) + \Theta(n)$.
We now use a recursion tree to estimate a solution to the above recurrence. The depth of the tree is dominated by the term $T(2n/3)$. Thus, a good estimate for the depth of the recursion tree is $O(\log_{3/2} n)$, which is $O(\lg n)$. And from the recurrence, it is clear that we do $O(n)$ at each level. Thus, an initial guess would be that the recurrence is bounded by $O(n \lg n)$. We shall now use the substitution method to verify our guess. Make an inductive hypothesis that $T(m) \le dm \lg m$, for all $m \in [1, n)$. We will, now, prove the hypothesis for $n$.

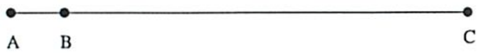
$$\begin{aligned}
T(n) &= d2n/3 \lg 2n/3 + dn/3 \lg n/3 + \Theta(n) \\
&= 2dn/3(\lg 2 + \lg n - \lg 3) + nd/3(\lg n - \lg 3) + \Theta(n) \\
&= dn/3(3 \lg n + 2 \lg 2 - \lg 3) + \Theta(n) \\
&= dn \lg n + dn/3(2 - 2 \lg 3) + \Theta(n) \\
&\le dn \lg n + 2dn/3(1 - \lg 3) + kn \quad \text{(from the definition of } \Theta(n)\text{)} \\
&= dn \lg n + n\,(2d/3(1 - \lg 3) + k) \\
&= dn \lg n - n\,(2d/3(\lg 3 - 1) - k) \\
&< dn \lg n, \quad \text{for all } d > 3k/2(\lg 3 - 1)
\end{aligned}$$

Thus, we have established that there is an absolute constant $d$ such that for sufficiently large $n$, $T(n) < dn \lg n$. Thus, $T(n) = O(n \lg n)$.

**Problem 3. SWAT Team.** [12 points]

Two swatsmen with fly swatters are located at arbitrary positions along a long corridor with many leaky windows. One at a time, houseflies appear at various locations along the corridor, and a swatsman goes to the location of the fly and swats it dead. The cost of a given strategy is the total distance traveled by the swatsmen. Argue that the greedy strategy of the closest swatsman going to the location of the fly is not $\alpha$-competitive for any finite $\alpha$.

(*Hint:* Consider the case that flies only appear at three locations $A$, $B$, and $C$, where $B$ falls between $A$ and $C$ and the distance from $A$ to $B$ is much smaller than the distance from $B$ to $C$, as shown below:



Consider the sequence $\langle C, A, B, A, B, A, B, A, B, A, \ldots \rangle$ of fly arrivals.)

**Solution:** Suppose on way of contradiction that there are $\alpha$ and $\beta$ such that $Greedy \le \alpha \cdot OPT + \beta$. Let $L$ be the distance between $A$ and $C$, and $\epsilon L$ be the distance between $A$ and $B$, where $0 < \epsilon < 1/2$. Let $T$ be the length of the sequence of flight arrivals. Take $T > (3\alpha/\epsilon) + (\beta/\epsilon L)$.
On the sequence of fly arrivals defined above:

- OPT sends one swatsman to $C$ for the first fly, and then, for the rest of the flies, send one swatsman to $A$ and one swatsman to $B$. The total cost is at most $3L$.
- The greedy algorithm, after sending one swatsman to $C$ and the other to $A$, lets the second swatsman take care of all flies in $B$ and $A$. The cost is at least $\epsilon LT$.

We chose the parameters so $\epsilon LT > \alpha \cdot 3L + \beta$, which contradicts the assumption.

**Problem 4. FIFO $= 2 \times$ LIFO.** [12 points]

A FIFO queue $Q$ supporting the operations ENQUEUE and DEQUEUE can be implemented using two stacks $S_1$ and $S_2$, each of which supports the operations PUSH, POP, and a test whether the stack is empty.

```
ENQUEUE($Q, x$):
1  PUSH($S_1, x$)
```

```
DEQUEUE($Q$):
1  if $S_1 = \emptyset$ and $S_2 = \emptyset$
2        error "queue underflow"
3  if $S_2 = \emptyset$
4        while $S_1 \neq \emptyset$
5              $x$ = POP($S_1$)
6              PUSH($S_2, x$)
7  return POP($S_2$)
```

Define a potential function $\Phi(Q) = c\,|S_1|$ for an appropriate constant $c > 0$, where $|S_1|$ is the number of items in $S_1$. Argue using a potential-function argument that each ENQUEUE and DEQUEUE operation takes $O(1)$ amortized time.

**Solution:**
Take the potential function $\Phi$ to be $3\,|S_1|$. Clearly, $\Phi$ is always non-negative and initially, $\Phi_0 = 0$. The amortized cost of each operation is its true cost plus the change in the potential. Let us evaluate it for each of the operations separately:

- ENQUEUE: The change in potential is $+3$. The true cost is 1. Thus, the amortized cost is at most 4.
- DEQUEUE: There are two cases:
  - If $S_2 \neq \emptyset$, the change in potential is 0, and the true cost is 3 (the operations in steps 1,3,7). Thus, the amortized cost is 3.
  - If $S_2 = \emptyset$, the change in potential is $-3\,|S_1|$, and the true cost is at most $3\,|S_1| + 4$. Thus, the amortized cost is at most 4.

In all cases, the amortized cost is at most 4.

**Problem 5. Big Edges.** [10 points]

Let $G = (V, E)$ be a connected undirected graph with distinct edge weights $w : E \to \mathbb{R}$, and let $c$ be a cycle in $G$. Consider the edge $e$ on $c$ with the largest weight, that is, $w(e) \geq w(e')$ for all $e' \in c$. Prove that $e$ does not belong to the minimum spanning tree of $G$.

**Solution:** Suppose for the sake of contradiction that $e = \{u, v\}$ is in a minimum spanning tree $T$ of $G$. If we remove $e$ from $T$, we divide it into two trees. This corresponds to a cut $(C, V - C)$ in $G$ such that $C$ is spanned by one of the trees and $V - C$ is spanned by the other. Since $e$ is in a cycle and it crosses this cut, there must exist another edge in the cycle, $e'$, that crosses this cut. (Any cycle must cross a cut an even number of times. To see why, follow the edges of the cycle.) Since edge $e'$ crosses the cut, adding it in will connect the two trees thus forming a spanning tree: $T - \{e\} \cup \{e'\}$.
Since all edge weights are distinct and $e$ is the edge with the largest weight on the cycle, necessarily $w(e') < w(e)$. The weight of $T'$ is therefore strictly lower than the weight of $T$, and therefore $T$ cannot be a minimum spanning tree.

**Problem 6. Minimum Madness.** [12 points] (4 parts)

Consider the following program to find the minimum value in an array $A$ of $n$ distinct elements.

```
MINIMUM(A, n):
1  min = ∞ // Set min to be a large value.
2  for i = 1 to n
3       if min > A[i]
4            min = A[i]
```

Assume that $A$'s elements are randomly permuted before invoking MINIMUM$(A, n)$ and that all permutations are equally likely. Let $X_i$ be the indicator random variable associated with the event that the variable $min$ is changed in line 4 during the $i$th iteration of the **for** loop, and let $Y = \sum_{i=1}^{n} X_i$ be the random variable denoting the total number of times $min$ is so updated.

**(a)** Argue that the probability that $A[i]$ is smaller than all the elements in $A[1 \,..\, i-1]$ is $1/i$.

**Solution:** Notice that the probability that $A[i]$ is smaller than all the elements $A[1 \,..\, i-1]$ is exactly equal to the probability that we find the minimum element of $A[1 \,..\, i]$ at position $i$ (here we use that all the elements are distinct). Since $A$ is randomly permuted and every permutation is equally likely, each one of positions $1 \,..\, i$ is equally likely to hold the minimum element, and the probability it lands at position $i$ is exactly $1/i$.

**(b)** Show that $\mathrm{E}[X_i] = 1/i$.

**Solution:**

$$\begin{aligned}\mathrm{E}[X_i] &= 0 \cdot \Pr\{X_i = 0\} + 1 \cdot \Pr\{X_i = 1\} \\ &= \Pr\{X_i = 1\} \\ &= 1/i\end{aligned}$$

**(c)** Prove that $\mathrm{E}[Y] = \Theta(\lg n)$.

**Solution:** By linearity of expectation and the sum of the harmonic series,

$$\begin{aligned}\mathrm{E}[Y] &= \mathrm{E}\left[\sum_{i=1}^{n} X_i\right] \\ &= \sum_{i=1}^{n} \mathrm{E}[X_i] \\ &= \sum_{i=1}^{n} 1/i \\ &= \ln n + O(1) \\ &= \lg n \cdot \lg e + O(1) \\ &= \Theta(\lg n)\end{aligned}$$

**(d)** Prove that for sufficiently large $n$, it holds that $\Pr\{Y \geq 5\mathrm{E}[Y]\} \leq 1/n^4$.

**Solution:** We use the following *multiplicative* version of Chernoff bound: For independent Boolean random variables $X_1, \ldots, X_n$, let $X = \sum_{i=1}^{n} X_i$, and $\mu = \mathrm{E}[X]$. Then, for any $0 < \delta \leq 2e - 1$,

$$\Pr\{X > (1+\delta)\mu\} < e^{-\mu\delta^2/4}.$$

For our problem, since $\mu > \ln n$, and so

$$\Pr\{Y > 5\mathrm{E}[Y]\} < e^{-4 \ln n} = \frac{1}{n^4}.$$

The statement of the Chernoff bound given in the beginning of the quiz is not strong enough to prove the required bound. We gave full credit to anyone who used it correctly.

# Practice Quiz 1

- The real Quiz 1 will be held on Thursday, October 15, in lecture.
- There will be a quiz review on Friday, October 9, during recitation.
- The quiz will be closed book. You may use one double sided Letter ($8\frac{1}{2}'' \times 11''$) or A4 crib sheet. No calculators or programmable devices are permitted.
- Write your solutions in the space provided. Extra scratch paper may be provided if you need more room, although your answer should fit in the given space.
- Do not waste time re-deriving facts that we have studied. It is sufficient to cite known results.
- Do not spend too much time on any one problem. Generally, a problem's point value is an indication of how much time to spend on it.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

**Problem 1. Recurrences** [15 points] (4 parts)

Solve the following recurrences by giving tight Θ-notation bounds. You do not need to justify your answers, but any justification that you provide will help when assigning partial credit. As usual, assume that for $n \leq 10$, $T(n) = O(1)$.

**(a)** [2 points] $T(n) = 3T(n/3) + 0.5n\lg(n)$.

**(b)** [4 points] $T(n) = 9T(\sqrt[3]{n}) + \Theta(\log(n))$.

**(c)** [4 points] $T(n) = T(2n/7) + T(5n/7) + \Theta(n)$.

**(d)** [5 points] Define $T(n)$ by the recursion $T(n) = 9(T(\lfloor n/3 \rfloor) - 1) + n^3 + 2n$ for $n \geq 1$, with base case $T(0) = 0$. Prove $T(n) \leq 3n^3/2$.

**Problem 2. True or False, and Justify** [13 points] (6 parts)

Circle **T** or **F** for each of the following statements, and briefly explain why. Your justification is worth more points than your true-or-false designation.

**(a) T F** [2 points] To achieve asymptotically optimal performance, a skip list must use promotion probability $p = 0.5$.

**(b) T F** [2 points] Universal hashing requires that you know what elements you'll hash in advance.

**(c) T F** [2 points] In a B-tree, the maximum number of children of an internal non-root node is at most twice the minimum of number of children.

**(d) T F** [2 points] A rotate operation on balanced tree always increases the depth of at least one node and decreases the depth of at least one node.

**(e) T F** [3 points] In a B-tree of minimum parameter $t$, every node contains at least $t - 1$ elements.

**Problem 3. Short Answer** [13 points] (4 parts)

Give *brief*, but complete, answers to the following questions.

**(a)** [2 points] What is the expected difference between the depth of the deepest leaf and the depth of the least deep leaf in a 2-3-4 tree containing $N$ elements?

**(b)** [3 points] Show how to find a divisor $d$ of $N$ such that $d$ is not 1 or $N$, given $x$, $y$ such that $x^2 = y^2 \bmod N$ and $x \neq y \bmod N$, $x \neq -y \bmod N$.

**(c)** [4 points] Let $\mathcal{H}$ be a universal hash family mapping $[1 \ldots N]$ to $[1 \ldots M]$. Let $X_{ijh}$ be the indicator variable for a collision between $i$ and $j$ under the hash function $h$, $i \neq j$ and $h \in \mathcal{H}$. What is $E(X_{ijh})$, where the expectation is taken over $i$, $j$, and $h$?

**(d)** [4 points] Consider a balanced binary tree of $n$ elements in which each node has an integer value. The weight of a path is the sum of the values of the nodes visited by the path. Give an optimal algorithm that computes the maximum possible weight of a path in the binary tree, starting at the root. What is the running time of your algorithm?

**Problem 4. Slightly-Longer Short Answer** [29 points] (5 parts)

Give *brief*, but complete, answers to the following questions.

**(a)** [5 points] A sequence of $n$ operations is performed, so that the $i^{th}$ operation costs $\lg(i)$ if $i$ is an exact power of 2, and 1 otherwise. That is the amortized cost per operation?

**(b)** [6 points] Define set $S = \{A \mid A = x^{(N-1)/2} \pmod{N}\}$ for a prime $N$. Is $S$ a sub-group of $Z_N^*$? If so, what can you say about the size of set $S$?

**(c)** [6 points] Given two sets $A$ and $B$ of $n$ integers, give an efficient deterministic algorithm to find $A \cap B$ and analyze its runtime. Can you do better with randomization? Explain.

**(d)** [6 points]

Consider an array $A$ of $n$ integers. Find all elements occurring at least $n/3$ times.

**(e)** [6 points]
Consider a sorted array $A$ of size $n$, containing distinct integers. Give an $O(\lg n)$ algorithm to find an index $i$ such that $A[i] = i$ (or *none*, if no such index exists). Does your algorithm still work if $A$ contains repeat elements? Explain why or why not.

**Problem 5. Searching in multiple lists** [8 points]

Consider two disjoint sorted arrays $A[1 \ldots m]$ and $B[1 \ldots n]$. Find an $O(\log k)$ time algorithm for computing the $k$-th smallest element in the union of the two arrays.

**Problem 6. The Eccentric Landlord** [8 points] (2 parts)

Your construction firm is hired to build an apartment building for an eccentric landlord. He wants his building to be a square of size $M \times M$, containing $M^2$ identical square apartments.

The landlord will add one tenant a day. When he can't fit a new tenant, he will tear down two sides of the building and have new walls built, expanding it an $(M+1) \times (M+1)$ building.

It costs your firm \$1 to build one apartment's exterior wall; your other costs (demolishing exterior walls, building interior walls, etc) are negligible. Your costs will be:

**Day 1:** \$4 (build four walls)

**Day 2:** \$6 (expand to 2x2)

**Day 3:** \$0 (tenant moves into empty unit)

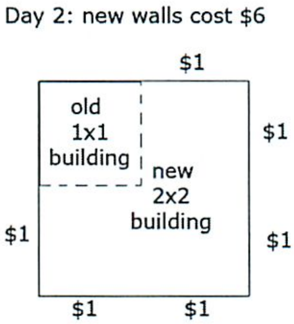
**Day 4:** \$0 (tenant moves into empty unit)

**Day 5:** \$8 (expand to 3x3)

**Day 6:** \$0 (tenant moves into empty unit)

⋮

The costs incurred on day 2 are shown below.



**(a)** [4 points] What will your asymptotic aggregate cost be for this project? Give your answer as a function of the number of days elapsed.

**(b)** [4 points] You convince the landlord to expand his building in bigger steps: Whenever he can't fit a new tenant, he will double the building side length (instead of increasing it by one unit). Repeat your analysis from part (a) for this new condition.

**Problem 7. Chemical testing** [15 points]

A chemistry lab is given $n$ samples, with the goal of determining which of the samples contain traces of a foreign substance. It is assumed that only few (say, at most $t$) samples test positive. The tests are very sensitive, and can detect even the slightest trace of the substance in a sample. However, each test is very expensive. Because of that, the lab decided to test "sample pools" instead. Each pool contains a mixture of some of the samples (each sample can participate in several pools). A test of a pool returns positive if any of the samples contributing to the pool contains a trace of the substance.

Design a testing method that correctly determines the positive samples using only $O(t \log n)$ tests. The method can be **adaptive**, i.e., the choice of the next test can depend on the outcomes of the previous tests.

*Design and Analysis of Algorithms* September 29, 2009
Massachusetts Institute of Technology 6.046J/18.410J
Profs. Erik Demaine and Shafi Goldwasser Handout 10

# Practice Quiz 1

- The real Quiz 1 will be held on Thursday, October 15, in lecture.
- There will be a quiz review on Friday, October 9, during recitation.
- The quiz will be closed book. You may use one double sided Letter ($8\frac{1}{2}'' \times 11''$) or A4 crib sheet. No calculators or programmable devices are permitted.
- Write your solutions in the space provided. Extra scratch paper may be provided if you need more room, although your answer should fit in the given space.
- Do not waste time re-deriving facts that we have studied. It is sufficient to cite known results.
- Do not spend too much time on any one problem. Generally, a problem's point value is an indication of how much time to spend on it.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

**Problem 1. Recurrences** [15 points] (4 parts)

Solve the following recurrences by giving tight $\Theta$-notation bounds. You do not need to justify your answers, but any justification that you provide will help when assigning partial credit. As usual, assume that for $n \leq 10$, $T(n) = O(1)$.

**(a)** [2 points] $T(n) = 3T(n/3) + 0.5n \lg(n)$.

**Solution:** Using case 2 of the Master's Method gives us $T(n) = \Theta(n \lg^2 n)$.

**(b)** [4 points] $T(n) = 9T(\sqrt[3]{n}) + \Theta(\log(n))$.

**Solution:** Let $n = 2^m$. Then the recurrence becomes $T(2^m) = 9T(2^{m/3}) + \Theta(m)$. Setting $S(m) = T(2^m)$ gives us $S(m) = 9S(m/3) + \Theta(m)$. Using case 1 of the Master Method gives us $S(m) = \Theta(m^2)$ or $T(n) = \Theta(\log^2 n)$

**(c)** [4 points] $T(n) = T(2n/7) + T(5n/7) + \Theta(n)$.

**Solution:** The Master Theorem doesn't apply here. Draw recursion tree. At each level, do $\Theta(n)$ work. Number of levels is $\log_{7/5} n = \Theta(\lg n)$, so guess $T(n) = \Theta(n \lg n)$ and use the substitution method to verify guess.

**(d)** [5 points] Define $T(n)$ by the recursion $T(n) = 9(T(\lfloor n/3 \rfloor) - 1) + n^3 + 2n$ for $n \geq 1$, with base case $T(0) = 0$. Prove $T(n) \leq 3n^3/2$.

**Solution:** Inductive hypothesis: $T(n) \leq 3n^3/2 - n$.
Base case: $T(0) = 0 \leq 30^3/2 - 0$.
Inductive step:

$$\begin{aligned} T(n) &= 9(T(\lfloor n/3 \rfloor) - 1) + n^3 + 2n \\ &\leq 9(3\lfloor n/3 \rfloor^3/2 - \lfloor n/3 \rfloor - 1) + n^3 + 2n \\ &\leq 9(3(n/3)^3/2 - n/3) + n^3 + 2n \\ &= 3n^3/2 - n. \end{aligned}$$

**Problem 2. True or False, and Justify** [13 points] (6 parts)

Circle **T** or **F** for each of the following statements, and briefly explain why. Your justification is worth more points than your true-or-false designation.

**(a) T F** [2 points] To achieve asymptotically optimal performance, a skip list must use promotion probability $p = 0.5$.

**Solution:** False. Any promotion probability between 0 and 1 achieves the same asymptotic performance.

**(b) T F** [2 points] Universal hashing requires that you know what elements you'll hash in advance.

**Solution:** False. Perfect hashing requires knowing the elements in advance. Universal hashing does not.

**(c) T F** [2 points] In a B-tree, the maximum number of children of an internal non-root node is at most twice the minimum of number of children.

**Solution:** True. For a B-tree with parameter $t$, there are at least $t$ and at most $2t$ children.

**(d) T F** [2 points] A rotate operation on balanced tree always increases the depth of at least one node and decreases the depth of at least one node.

**Solution:** TRUE. Every rotate operation demotes the root of a subtree and promotes a new node to that position. Promotion decreases a node's depth. See CLRS 13.2 or Lecture 7 for a description/illustration of rotation.

**(e) T F** [3 points] In a B-tree of minimum parameter $t$, every node contains at least $t - 1$ elements.

**Solution:** FALSE, Normally nodes in a B-tree of parameter $t$ must have between $t - 1$ and $2t - 1$ elements, but the root node is exempted from this rule to accomodate trees with fewer than $t - 1$ elements. Consider a B-tree of parameter 3 that contains one element: the root node contains 1 element, but here $t - 1 = 2$.

**Problem 3. Short Answer** [13 points] (4 parts)

Give *brief*, but complete, answers to the following questions.

**(a)** [2 points] What is the expected difference between the depth of the deepest leaf and the depth of the least deep leaf in a 2-3-4 tree containing $N$ elements?

**Solution:** Zero. All leaves are at the same level.

**(b)** [3 points] Show how to find a divisor $d$ of $N$ such that $d$ is not 1 or $N$, given $x$, $y$ such that $x^2 = y^2 \bmod N$ and $x \neq y \bmod N$, $x \neq -y \bmod N$.

**Solution:** Compute $\gcd(x - y, N)$ or compute $\gcd(x + y, N)$

**(c)** [4 points] Let $\mathcal{H}$ be a universal hash family mapping $[1 \ldots N]$ to $[1 \ldots M]$. Let $X_{ijh}$ be the indicator variable for a collision between $i$ and $j$ under the hash function $h$, $i \neq j$ and $h \in \mathcal{H}$. What is $E(X_{ijh})$, where the expectation is taken over $i$, $j$, and $h$?

**Solution:** By the definition of a universal hash family, the probability of a collision is $1/M$, regardless of $i$ and $j$. So the expected value of the indicator variable is $1/M$.

**(d)** [4 points] Consider a balanced binary tree of $n$ elements in which each node has an integer value. The weight of a path is the sum of the values of the nodes visited by the path. Give an optimal algorithm that computes the maximum possible weight of a path in the binary tree, starting at the root. What is the running time of your algorithm?

**Solution:** If the tree consists of a single node (ie, we are at a leaf), then the answer is simply the weight of that node. Otherwise, we recurse using

$$\textsc{MaxPath}(\mathit{root}) = w(\mathit{root}) + \max\left\{ \begin{array}{l} \textsc{MaxPath}(\mathit{root} \rightarrow \mathit{left}), \\ \textsc{MaxPath}(\mathit{root} \rightarrow \mathit{right}) \end{array} \right\}$$

This algorithm accesses each node exactly once, so the runtime is $\Theta(n)$.

**Problem 4. Slightly-Longer Short Answer** [29 points] (5 parts)

Give *brief*, but complete, answers to the following questions.

**(a)** [5 points] A sequence of $n$ operations is performed, so that the $i^{th}$ operation costs $\lg(i)$ if $i$ is an exact power of 2, and 1 otherwise. That is the amortized cost per operation?

**Solution:** True. Let $c(i)$ be the cost of the $i^{th}$ operation

$$c(i) = \begin{cases} \lg i & \text{if } i = 2^k, k \text{ integer} \\ 1 & \text{otherwise} \end{cases}$$

For any $n$, the total cost of $n$ operations is

$$\begin{aligned} \sum_{i=1}^{n} c(i) &= n - \lfloor \lg n \rfloor + \sum_{i=1}^{\lfloor \lg n \rfloor} i \\ &= n + \Theta(\lg^2(n)) = \Theta(n) \end{aligned}$$

Therefore, the amortized cost per operation is $\Theta(1)$.

**(b)** [6 points] Define set $S = \{A \in Z_N^* \mid A = x^{(N-1)/2} \pmod{N}\}$ for a prime $N$. Is $S$ a sub-group of $Z_N^*$? If so, what can you say about the size of set $S$?

**Solution:** $S$ is a sub-group of $Z_N^*$:

- identity: $1 \in S$
- closure: given $A = x^{(N-1)/2} \pmod{N}$ and $B = y^{(N-1)/2} \pmod{N}$, $AB = (xy)^{(N-1)/2} \pmod{N} \in S$
- inverse: given $A = x^{(N-1)/2} \pmod{N}$, since $x \neq 0$ and $N$ is a prime, it must be that $x \in Z_N^*$. Thus, $x$ must have an inverse, so $A^{-1} = (x^{-1})^{(N-1)/2} \pmod{N}$, which is also in $S$.

By Lagrange's Theorem, we know that $|S|$ divides $|Z_N^*|$, but in this case, we can say something stronger: $A = x^{(N-1)/2} \pmod{N}$, so $A^2 = x^{(N-1)} = 1 \pmod{N}$ (by Fermat's Little Theorem), which has only two solutions (1 and $N-1$ by Modular Sqrt Theorem), so $|S| = 2$.

**(c)** [6 points] Given two sets $A$ and $B$ of $n$ integers, give an efficient deterministic algorithm to find $A \cap B$ and analyze its runtime. Can you do better with randomization? Explain.

**Solution:** For the deterministic algorithm, sort each list. Then, iterate through the elements looking for elements common to both arrays. This takes $O(n \lg n)$ time. Using randomization, hash the elements of $A$. Then iterate through $B$, looking up elements in the hash of $A$. The expected running time is $O(n)$.

**(d)** [6 points]

Consider an array $A$ of $n$ integers. Find all elements occurring at least $n/3$ times.

**Solution:** Replace the $i$th element with a pair $(A[i], i)$ to make them all distinct. Comparison between pairs is done by comparing the first elements and breaking ties by the second elements.

Use the select algorithm to find the elements of ranks $n/3$, $2n/3$ and $n$. If an element occurs at least $n/3$ times, it must be one of those three elements. Check all three to see if any of them occurs at least $n/3$ times. The running time is O(n).

**(e)** [6 points]

Consider a sorted array $A$ of size $n$, containing distinct integers. Give an $O(\lg n)$ algorithm to find an index $i$ such that $A[i] = i$ (or *none*, if no such index exists). Does your algorithm still work if $A$ contains repeat elements? Explain why or why not.

**Solution:** Consider $A[n/2]$. If $A[n/2] = n/2$, then we're done. Otherwise, if $A[n/2] > n/2$, recurse on $A[1 \ldots n/2 - 1]$. If $A[n/2] < n/2$, recurse on $A[n/2+1 \ldots n]$. The runtime is $O(\lg n)$.

If there are repeat elements, then we can no longer ensure that the answer is on one side of the median, since it may be true that $A[1] = 1$ and $A[n] = n$, for any value of the median between 2 and $n - 1$.

**Problem 5. Searching in multiple lists** [8 points]

Consider two disjoint sorted arrays $A[1 \ldots m]$ and $B[1 \ldots n]$. Find an $O(\log k)$ time algorithm for computing the $k$-th smallest element in the union of the two arrays.

**Solution:** Consider $A[k/2]$ and $B[k/2]$. Without loss of generality, assume $A[k/2] < B[k/2]$. Then $A[k/2]$ is greater than at most $k$ elements. Furthermore the elements $A[1 \ldots k/2 - 1]$ are all less than the $k$-th element, so we can eliminate them. Similarly, $B[k/2]$ is greater than at least $k$ elements, so the elements $B[k/2 + 1 \ldots n]$ are all larger than the $k$-th element. We can therefore eliminate them too. We are therefore left with two subarrays, and we now want to find the $k/2$-th element (since we eliminated $k/2$ elements that were guaranteed to be less than the $k$-th element). This divide-and-conquer algorithm follows the recursion $T(k) = T(k/2) + 1$, which is $O(\log k)$.

**Problem 6. The Eccentric Landlord** [8 points] (2 parts)

Your construction firm is hired to build an apartment building for an eccentric landlord. He wants his building to be a square of size $M \times M$, containing $M^2$ identical square apartments.

The landlord will add one tenant a day. When he can't fit a new tenant, he will tear down two sides of the building and have new walls built, expanding it an $(M + 1) \times (M + 1)$ building.

It costs your firm $1 to build one apartment's exterior wall; your other costs (demolishing exterior walls, building interior walls, etc) are negligible. Your costs will be:

**Day 1:** $4 (build four walls)

**Day 2:** $6 (expand to 2x2)

**Day 3:** $0 (tenant moves into empty unit)

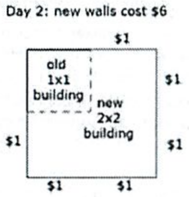
**Day 4:** $0 (tenant moves into empty unit)

**Day 5:** $8 (expand to 3x3)

**Day 6:** $0 (tenant moves into empty unit)

⋮

The costs incurred on day 2 are shown below.



(a) [4 points] What will your asymptotic aggregate cost be for this project? Give your answer as a function of the number of days elapsed.

**Solution:** We build between 1 and 2 new walls for each tenant, and tenants arrive at a rate of 1/day, so the cost per day is $O(1)$. The aggregate cost is therefore $O(n)$.

(b) [4 points] You convince the landlord to expand his building in bigger steps: Whenever he can't fit a new tenant, he will double the building side length (instead of increasing it by one unit). Repeat your analysis from part (a) for this new condition.

**Solution:** We must spend $O(\sqrt{n})$ dollars to fit $O(n)$ tenants, since the cost is dominated by the most current addition.

**Problem 7. Chemical testing** [15 points]

A chemistry lab is given $n$ samples, with the goal of determining which of the samples contain traces of a foreign substance. It is assumed that only few (say, at most $t$) samples test positive. The tests are very sensitive, and can detect even the slightest trace of the substance in a sample. However, each test is very expensive. Because of that, the lab decided to test "sample pools" instead. Each pool contains a mixture of some of the samples (each sample can participate in several pools). A test of a pool returns positive if any of the samples contributing to the pool contains a trace of the substance.

Design a testing method that correctly determines the positive samples using only $O(t \log n)$ tests. The method can be **adaptive**, i.e., the choice of the next test can depend on the outcomes of the previous tests.

**Solution:** There exist several related algorithms that solve this problem. The simplest one proceeds as follows: we divide the samples into $2t$ groups of size $\frac{n}{2t}$ each. We pool and test each group. Since at most $t$ groups are positive, we can label at least $n/2$ samples as negative. Then we recurse on the remaining $n/2$ samples. It is easy to see that the number of recursion levels is $O(\log n)$. Since $2t$ tests are performed at each level, the total number of tests is at most $O(t \log n)$.

A different algorithm divide the samples into two groups of size $n/2$. Both groups are tested, and the algorithm recurses on group(s) that test positive. As before, the recursion tree has depth $\log n$, since we divide the group size by 2 at each level. Moreover, the recursion tree contains at most $t$ leaves. Therefore the total number of tree nodes (and therefore tests) is $O(t \log n)$.

*Design and Analysis of Algorithms* | March 3, 2008
Massachusetts Institute of Technology | 6.046J/18.410J
Prof. Manolis Kellis and Dr. Marten van Dijk | Practice Quiz 1

# Practice Quiz 1

- Quiz 1 will be held on Tuesday, March 10, in lecture.
- There will be a quiz review on Thursday from 6-8pm, Location TBD. Keep an eye out for the announcement.
- The quiz will consist of several multi-part problems. You will have 80 minutes to earn 80 points. (This practice quiz contains 120 points worth of questions. The real quiz will be shorter.)
- The quiz is closed book. You may bring one double sided Letter ($8\frac{1}{2}'' \times 11''$) or A4 crib sheet. No calculators or programmable devices are permitted.
- Write your solutions in the space provided. Although extra scratch paper will be provided, your answer should fit in the given space.
- Do not waste time and paper re-deriving facts that we have studied. It is sufficient to cite known results.
- Do not spend too much time on any one problem. Read them all through first, and attack them in the order that allows you to make the most progress. Generally, a problem's point value is an indication of how much time to spend on it.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

| Problem | Points | Grade | Initials |
|---|---|---|---|
| 1 | 2 | | |
| 2 | 8 | | |
| 3 | 40 | | |
| 4 | 15 | | |
| 5 | 15 | | |
| 6 | 20 | | |
| 7 | 20 | | |
| Total | 120 | | |

**Name:** ____________________

Circle your recitation instructor:

Matthew Webber (F 11, F1)　　Kevin Matulef (F2)　　Huy Nguyen (F3)

**Problem 1.** [2 points] Write your name on every page!

**Problem 2. Recurrences** [8 points] (2 parts)

Solve the following recurrences by giving tight $\Theta$-notation bounds. You do not need to justify your answers, but any justification that you provide will help when assigning partial credit. As usual, assume that for $n \leq 10$, $T(n) = O(1)$.

**(a)** [4 points] $T(n) = 9T(\sqrt[3]{n}) + \Theta(\log(n))$.

**(b)** [4 points] $T(n) = T(2n/7) + T(5n/7) + \Theta(n)$.

**Problem 3. True or False, and Justify** [40 points] (8 parts)

Circle **T** or **F** for each of the following statements, and briefly explain why. The better your argument, the higher your grade, but be brief. Your justification is worth more points than your true-or-false designation.

**(a) T F** [5 points] Suppose that $H$ is a finite family of universal hash functions of range (table) size 999. $|H|$ is divisible by 9.

**(b) T F** [5 points] Let $H$ be a family of universal hash functions that map the universe $K$ of keys into the range $\{0, 1, \cdots, n-1\}$. For a given $x \in K$, and $h$ is a function chosen randomly from $H$, $Pr[h(x) = 0] = Pr[h(x) = 1] = \cdots = Pr[h(x) = n-1] = \frac{1}{n}$.

**(c) T F** [5 points] A rotate operation on balanced tree always increases the depth of at least one node and decreases the depth of at least one node.

**(d) T F** [5 points] In a B-tree of minimum parameter $t$, every node contains at least $t - 1$ elements.

**(e) T F** [5 points] The number of leaves (leaf nodes) in every B-tree is at least $1/2$ the total number of nodes.

**(f) T F** [5 points] The difference between the depth of the deepest and least deep node in a 2-3-4 tree is $\Theta(\log(n))$, where $n$ is the number of nodes in the tree.

**(g) T F** [5 points] Consider a dynamic table that doubles in size when an insert operation causes the table to overflow, and halves when a delete operation causes the table to be less than 1/4 full. If we assign an amortized cost of 4 per insert (with deletes free), then for every sequence of $n$ consecutive operations, amortized costs serve as an upper bound on true costs.

**(h) T F** [5 points] A sequence of $n$ operations is performed, so that the $i^{th}$ operation costs $\lg(i)$ if $i$ is an exact power of 2, and 1 otherwise. Then the amortized cost per operation is $\Theta(1)$.

**Problem 4. Polynomial Interpolation** [15 points] Suppose you are given numbers $r_1, r_2, \cdots, r_n$ and want to compute the coefficients of the degree $n$ polynomial with exactly those roots, i.e. $\prod_{i=1}^{n}(x - r_i)$. Give an $O(n \log^2 n)$ algorithm.

**Problem 5. Chemical testing** [15 points]

A chemistry lab is given $n$ samples, with the goal of determining which of the samples contain traces of a foreign substance. It is assumed that only few (say, at most $t$) samples test positive. The tests are very sensitive, and can detect even the slightest trace of the substance in a sample. However, each test is very expensive. Because of that, the lab decided to test "sample pools" instead. Each pool contains a mixture of some of the samples (each sample can participate in several pools). A test of a pool returns positive if any of the samples contributing to the pool contains a trace of the substance.

Design a testing method that correctly determines the positive samples using only $O(t \log n)$ tests. The method can be **adaptive**, i.e., the choice of the next test can depend on the outcomes of the previous tests.

**Problem 6. Two-array hashing** [20 points]

Alyssa P. Hacker runs an internet company that sells $n$ different products. In order to quickly access information about the $n$ products for sale, each product is hashed to a size-$n$ hash table using a simple uniform hash function, with collisions resolved by chaining. Alyssa is happy with this approach because in expectation, a query takes $O(1)$ time. However, the downside of the approach is that it is quite likely some slot of the table will have many items hashing to it.[1]

To solve this problem, Alyssa comes up with an idea for ***two-array hashing***, which is defined as follows. Given $n$ items, allocate *two* arrays $A_1$ and $A_2$, each of size $n^{1.5}$. When inserting a new item, map it to one slot in each of the arrays using two different simple uniform hash functions $h_1$ and $h_2$. Place the item only in the less crowded of the two slots. We say that a ***collision*** occurs if both of the two slots are already nonempty.

**(a)** [8 points] Consider the $k$th items inserted into the two-array hash table. Let $C_k$ be an indicator random variable with

$$C_k = \begin{cases} 1 & : \text{ if the } k\text{th insert causes a collision} \\ 0 & : \text{ otherwise.} \end{cases}$$

Show that $E[C_k] \leq (k-1)^2/n^3$.

[1] One can show that with high probability some slot of the table has $\Theta(\log n/\log\log n)$ items hashing to it, but you do not need to know or be able to prove this fact.

**(b)** [12 points] Define the random variable $C = \sum_{k=1}^{n} C_k$. What does the variable $C$ represent? Show that $E[C] = O(1)$, and conclude that the fullest slot in the hash table contains $O(1)$ items in expectation.

**Problem 7. Broadcast channel** [20 points]

A set of up to $n$ processors attempt to communicate over a network. The communication process is deemed successful if *any* of the processors manages to broadcast its information (since the successful processor can then lead the remainder of the communication process). However, the only means of communication is through a common broadcast channel. At any given time step (we assume the time is discrete), any subset of the processors can attempt to communicate through the channel by sending a message. The channel operates as follows:

- If *none* of the processors attempts to send a message, then all processors receive a special "none" message.
- If *only one* of the processors attempts to send a message, then all processors receive that message, and the communication process is deemed successful.
- If *two or more* processors attempt to send a message, then all processors receive a special "collision" message.

Suppose that the number of processors is at least $n/2$. Design a randomized protocol that, if followed by all processors, will result in successful communication. The expected number of time steps used by the protocol should be $O(1)$.

You can assume all processors know the upper bound $n$ and the lower bound $n/2$.

# SCRATCH PAPER

SCRATCH PAPER

*Design and Analysis of Algorithms*
Massachusetts Institute of Technology
Prof. Manolis Kellis and Dr. Marten van Dijk

March 3, 2008
6.046J/18.410J
Practice Quiz 1

# Practice Quiz 1

- Quiz 1 will be held on Tuesday, March 10, in lecture.
- There will be a quiz review on Thursday from 6-8pm, Location TBD. Keep an eye out for the announcement.
- The quiz will consist of several multi-part problems. You will have 80 minutes to earn 80 points. (This practice quiz contains 120 points worth of questions. The real quiz will be shorter.)
- The quiz is closed book. You may bring one double sided Letter ($8\frac{1}{2}'' \times 11''$) or A4 crib sheet. No calculators or programmable devices are permitted.
- Write your solutions in the space provided. Although extra scratch paper will be provided, your answer should fit in the given space.
- Do not waste time and paper re-deriving facts that we have studied. It is sufficient to cite known results.
- Do not spend too much time on any one problem. Read them all through first, and attack them in the order that allows you to make the most progress. Generally, a problem's point value is an indication of how much time to spend on it.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

| Problem | Points | Grade | Initials |
|---|---|---|---|
| 1 | 2 | | |
| 2 | 8 | | |
| 3 | 40 | | |
| 4 | 15 | | |
| 5 | 15 | | |
| 6 | 20 | | |
| 7 | 20 | | |
| Total | 120 | | |

**Name:** ________________________________

Circle your recitation instructor:

Matthew Webber (F 11, F1) Kevin Matulef (F2) Huy Nguyen (F3)

**Problem 1.** [2 points] Write your name on every page!

**Problem 2. Recurrences** [8 points] (2 parts)

Solve the following recurrences by giving tight $\Theta$-notation bounds. You do not need to justify your answers, but any justification that you provide will help when assigning partial credit. As usual, assume that for $n \leq 10$, $T(n) = O(1)$.

**(a)** [4 points] $T(n) = 9T(\sqrt[3]{n}) + \Theta(\log(n))$.

**Solution:** Let $n = 2^m$. Then the recurrence becomes $T(2^m) = 9T(2^{m/3}) + \Theta(m)$. Setting $S(m) = T(2^m)$ gives us $S(m) = 9S(m/3) + \Theta(m)$. Using case 1 of the Master Method gives us $S(m) = \Theta(m^2)$ or $T(n) = \Theta(\log^2 n)$

**(b)** [4 points] $T(n) = T(2n/7) + T(5n/7) + \Theta(n)$.

**Solution:** The Master Theorem doesn't apply here. Draw recursion tree. At each level, do $\Theta(n)$ work. Number of levels is $\log_{7/5} n = \Theta(\lg n)$, so guess $T(n) = \Theta(n \lg n)$ and use the substitution method to verify guess.

**Problem 3. True or False, and Justify** [40 points] (8 parts)

Circle **T** or **F** for each of the following statements, and briefly explain why. The better your argument, the higher your grade, but be brief. Your justification is worth more points than your true-or-false designation.

**(a) T F** [5 points] Suppose that $H$ is a finite family of universal hash functions of range (table) size 999. $|H|$ is divisible by 9.

**Solution:** False. Let $h$ be the identical function that maps the key universe $\{1, 2, \cdots, 999\}$ into the range (table) $\{1, 2, \cdots, 999\}$ such that $h(x) = x$. Then the hashing function $h$ does not have any collision. $H = \{h\}$ is a universal hashing family since for any $x \neq y$, $Pr[h(x) = h(y)] = 0 < 1/999$. $|H| = 1$ is not divisible by 9.

Note: In this question, we also gave credit to those who used the following definition of universal hashing: "$H$ is a universal hashing family if for any $x \neq y$ and a hash function $h$ chosen randomly from $H$, the probability that $h(x) = h(y)$ is **equal to** $1/m$" (although in the correct definition, this probability is **at most** $1/m$). The answer is then *True* since for any pair $x \neq y$, $|\{h \in H | h(x) = h(y)\}| = |H|/999$ is an integer, therefore, $|H|$ is divisible by 999.

**(b) T F** [5 points] Let $H$ be a family of universal hash functions that map the universe $K$ of keys into the range $\{0, 1, \cdots, n-1\}$. For a given $x \in K$, and $h$ is a function chosen randomly from $H$, $Pr[h(x) = 0] = Pr[h(x) = 1] = \cdots = Pr[h(x) = n-1] = \frac{1}{n}$.

**Solution:** False.
Let $\mathcal{H} = \{h_1, h_2, h_3\}$, where the three hash functions map the universe $\{A, B, C, D\}$ of keys into the range $\{0, 1, 2\}$ according to the following table:

| $x$ | $h_1(x)$ | $h_2(x)$ | $h_3(x)$ |
|---|---|---|---|
| $A$ | 1 | 0 | 2 |
| $B$ | 0 | 1 | 2 |
| $C$ | 0 | 0 | 0 |
| $D$ | 1 | 1 | 0 |

$H$ is a universal hashing family. However, for $x = C$ and any $h \in H$, $Pr[h(x) = 0] = 1 > Pr[h(x) = 1] = Pr[h(x) = 2] = 0$.

**(c) T F** [5 points] A rotate operation on balanced tree always increases the depth of at least one node and decreases the depth of at least one node.

**Solution:** TRUE. Every rotate operation demotes the root of a subtree and promotes a new node to that position. Promotion decreases a node's depth. See CLRS 13.2 or Lecture 7 for a description/illustration of rotation.

**(d) T F** [5 points] In a B-tree of minimum parameter $t$, every node contains at least $t - 1$ elements.

**Solution:** FALSE, Normally nodes in a B-tree of parameter $t$ must have between $t - 1$ and $2t - 1$ elements, but the root node is exempted from this rule to accomodate trees with fewer than $t - 1$ elements. Consider a B-tree of parameter 3 that contains one element: the root node contains 1 element, but here $t - 1 = 2$.

**(e) T F** [5 points] The number of leaves (leaf nodes) in every B-tree is at least $1/2$ the total number of nodes.

**Solution:** TRUE. Every non-leaf in a B-tree must have at least 2 children. The number of nodes in the level above the leaves is at most $1/2$ the number of leaves. The number of nodes in the level above that is at most $1/4$ the number of leaves, and so on up the tree. This sum cannot exceed the total number of leaves.

**(f) T F** [5 points] The difference between the depth of the deepest and least deep node in a 2-3-4 tree is $\Theta(\log(n))$, where $n$ is the number of nodes in the tree.

**Solution:** True. The difference in depths is one less than the height of the tree, so it is $\Theta(\log(n))$.

**(g) T F** [5 points] Consider a dynamic table that doubles in size when an insert operation causes the table to overflow, and halves when a delete operation causes the table to be less than 1/4 full. If we assign an amortized cost of 4 per insert (with deletes free), then for every sequence of $n$ consecutive operations, amortized costs serve as an upper bound on true costs.

**Solution:** False. As an example, consider the cost of inserting 17 elements and then deleting those 17 elements. The cost is

- 1*17 for the inserts.
- 1*17 for the deletes.
- 1+2+4+8+16 = 31 for table expansions.
- 7+3+1 = 11 for table contractions.
- Total = 17+17+31+11 = 76.

Note: Due to the subtleties of the problem, we decided to disregard this question and give everyone 5 points.

**(h) T F** [5 points] A sequence of $n$ operations is performed, so that the $i^{th}$ operation costs $\lg(i)$ if $i$ is an exact power of 2, and 1 otherwise. Then the amortized cost per operation is $\Theta(1)$.

**Solution:** True. Let $c(i)$ be the cost of the $i^{th}$ operation

$$c(i) = \begin{cases} \lg i & \text{if } i = 2^k, k \text{ integer} \\ 1 & \text{otherwise} \end{cases}$$

For any $n$, the total cost of $n$ operations is

$$\begin{aligned} \sum_{i=1}^{n} c(i) &= n - \lfloor \lg n \rfloor + \sum_{i=1}^{\lfloor \lg n \rfloor} i \\ &= n + \Theta(\lg^2(n)) = \Theta(n) \end{aligned}$$

Therefore, the amortized cost per operation is $\Theta(1)$.

**Problem 4. Polynomial Interpolation** [15 points] Suppose you are given numbers $r_1, r_2, \cdots, r_n$ and want to compute the coefficients of the degree $n$ polynomial with exactly those roots, i.e. $\prod_{i=1}^{n}(x - r_i)$. Give an $O(n \log^2 n)$ algorithm.

**Solution:** A simple recursive algorithm suffices. Compute $\prod_{i=1}^{\lfloor n/2 \rfloor}(x - r_i)$ and $\prod_{i=\lfloor n/2 \rfloor+1}^{n}(x - r_i)$ and then multiply the two degree-$n/2$ polynomials together using fast polynomial multiplication via the FFT. The base case is clearly constant time and the combine step takes $O(n \log n)$ time. Thus the running time is given by the recurrence $T(n) = 2T(n/2) + \Theta(n \log n)$. By case 2 of the generalized Master Theorem (as in the lecture), $T(n)$ is $O(n \log^2 n)$

**Problem 5. Chemical testing** [15 points]

A chemistry lab is given $n$ samples, with the goal of determining which of the samples contain traces of a foreign substance. It is assumed that only few (say, at most $t$) samples test positive. The tests are very sensitive, and can detect even the slightest trace of the substance in a sample. However, each test is very expensive. Because of that, the lab decided to test "sample pools" instead. Each pool contains a mixture of some of the samples (each sample can participate in several pools). A test of a pool returns positive if any of the samples contributing to the pool contains a trace of the substance.

Design a testing method that correctly determines the positive samples using only $O(t \log n)$ tests. The method can be **adaptive**, i.e., the choice of the next test can depend on the outcomes of the previous tests.

**Solution:** There exist several related algorithms that solve this problem. The simplest one proceeds as follows: we divide the samples into $2t$ groups of size $\frac{n}{2t}$ each. We pool and test each group. Since at most $t$ groups are positive, we can label at least $n/2$ samples as negative. Then we recurse on the remaining $n/2$ samples. It is easy to see that the number of recursion levels is $O(\log n)$. Since $2t$ tests are performed at each level, the total number of tests is at most $O(t \log n)$.

A different algorithm divide the samples into two groups of size $n/2$. Both groups are tested, and the algorithm recurses on group(s) that test positive. As before, the recursion tree has depth $\log n$, since we divide the group size by 2 at each level. Moreover, the recursion tree contains at most $t$ leaves. Therefore the total number of tree nodes (and therefore tests) is $O(t \log n)$.

**Problem 6. Two-array hashing** [20 points]

Alyssa P. Hacker runs an internet company that sells $n$ different products. In order to quickly access information about the $n$ products for sale, each product is hashed to a size-$n$ hash table using a simple uniform hash function, with collisions resolved by chaining. Alyssa is happy with this approach because in expectation, a query takes $O(1)$ time. However, the downside of the approach is that it is quite likely some slot of the table will have many items hashing to it.[1]

To solve this problem, Alyssa comes up with an idea for ***two-array hashing***, which is defined as follows. Given $n$ items, allocate *two* arrays $A_1$ and $A_2$, each of size $n^{1.5}$. When inserting a new item, map it to one slot in each of the arrays using two different simple uniform hash functions $h_1$ and $h_2$. Place the item only in the less crowded of the two slots. We say that a ***collision*** occurs if both of the two slots are already nonempty.

**(a)** [8 points] Consider the $k$th items inserted into the two-array hash table. Let $C_k$ be an indicator random variable with

$$C_k = \begin{cases} 1 & : \text{ if the } k\text{th insert causes a collision} \\ 0 & : \text{ otherwise.} \end{cases}$$

Show that $E[C_k] \le (k-1)^2/n^3$.

**Solution:** The $k$th insert causes a collision if and only if $h_1(k)$ is a nonempty slot of $A_1$ and $h_2(k)$ is a nonempty slot of $A_2$. At the time of the $k$th insert, both $A_1$ and $A_2$ have at most $k-1$ nonempty slots (possibly fewer). Since we are assuming $h_1$ and $h_2$ hash uniformly and independently, the probability that they both hash to nonempty slots is at most $(k-1)/n^{1.5} \cdot (k-1)/n^{1.5} = (k-1)^2/n^3$.

Partial credit was given for correctly bounding the probability of the $k$th item hashing to a nonempty slot of a single array as $\le (k-1)/n^{1.5}$. Other attempts at a solution received a small number of points.

[1] One can show that with high probability some slot of the table has $\Theta(\log n/\log\log n)$ items hashing to it, but you do not need to know or be able to prove this fact.

**(b)** [12 points] Define the random variable $C = \sum_{k=1}^{n} C_k$. What does the variable $C$ represent? Show that $E[C] = O(1)$, and conclude that the fullest slot in the hash table contains $O(1)$ items in expectation.

**Solution:** The variable $C$ represents the number of elements that collide with other elements (note this is slightly different than the number of pairs of elements that collide).
To solve for $E[C]$, we just need linearity of expectation and part (a) above:

$$\begin{aligned} E[C] &= E\left[\sum_{k=1}^{n} C_k\right] \\ &= \sum_{k=1}^{n} E[C_k] \\ &\le \sum_{k=1}^{n} \frac{(k-1)^2}{n^3} \\ &\le \frac{n(n-1)^2}{n^3} \\ &\le 1 \end{aligned}$$

The expected size of the fullest slot in the hash table is upper bounded by the expected number of elements that collide with other elements. Since the latter is bounded by a constant, so is the former. Thus Alyssa's scheme works.

To receive full credit on this problem it should have been made clear that linearity of expectation was being used. Sloppiness here usually received a minor point deduction.

**Problem 7. Broadcast channel** [20 points]

A set of up to $n$ processors attempt to communicate over a network. The communication process is deemed successful if *any* of the processors manages to broadcast its information (since the successful processor can then lead the remainder of the communication process). However, the only means of communication is through a common broadcast channel. At any given time step (we assume the time is discrete), any subset of the processors can attempt to communicate through the channel by sending a message. The channel operates as follows:

- If *none* of the processors attempts to send a message, then all processors receive a special "none" message.
- If *only one* of the processors attempts to send a message, then all processors receive that message, and the communication process is deemed successful.
- If *two or more* processors attempt to send a message, then all processors receive a special "collision" message.

Suppose that the number of processors is at least $n/2$. Design a randomized protocol that, if followed by all processors, will result in successful communication. The expected number of time steps used by the protocol should be $O(1)$.

You can assume all processors know the upper bound $n$ and the lower bound $n/2$.

**Solution:** We assume $n > 1$, since otherwise the problem is trivial. The algorithm is as follows: at each time step, each processor sends its message with probability $1/n$. If exactly one of the processors manages to broadcast its message, the whole process stops. Otherwise, the protocol is repeated in the next time step.

The analysis is as follows: we will prove that the expected number of time steps used by the protocol is constant. Since each processor sends a message with probability $1/n$, the number of messages sent in each time step follows the binomial distribution. In particular, if there are $k$ processors, the probability that exactly one message is sent at a given time step is

$$P = \binom{k}{1}\frac{1}{n}(\frac{n-1}{n})^{k-1} \geq 1/2 \cdot (1-1/n)^{k-1} \geq 1/2 \cdot (1-1/n)^n,$$

Since $(1-1/n)^n \to 1/e$ as $n \to \infty$ and $(1-1/n)^n > 0$ for $n > 1$, it follows that $(1-1/n)^n \geq \delta$ for some absolute constant $\delta > 0$ (in fact, we can take $\delta = 1/4$). This implies that $P \geq \delta/2$. The expected number of time steps used by the protocol is at most $1/P \leq 2/\delta = O(1)$.

Some students proposed related randomized algorithms, with running times of $n$ steps or more. We gave partial credit for those. Also, some students observed that the above procedure guarantees that the expected number of processors that broadcast at each step is $O(1)$. This is correct, but not sufficient to give an $O(1)$ bound for the expected number of *steps*. Again, partial credit was given.

SCRATCH PAPER

SCRATCH PAPER

*Introduction to Algorithms* March 4, 2008
Massachusetts Institute of Technology 6.046J/18.410J
Professors Manolis Kellis and Piotr Indyk Practice Quiz 1

# Practice Quiz 1

**Problem 1. Algorithms and running times** (5 parts) [5 points]

Match each algorithm below with the tightest asymptotic upper bound for its worst-case running time by inserting one of the letters A, B, . . ., E into the corresponding box. **Some running times may be used multiple times or not at all.** For sorting algorithms, $n$ is the number of input elements. For matrix algorithms, the input matrix has size $n \times n$.

You need not justify your answers. Because points will be deducted for wrong answers, do not guess unless you are reasonably sure.

| | | |
|---|---|---|
| □ | Insertion sort | A: $O(\lg n)$ |
| □ | Binary Search | B: $O(n)$ |
| □ | BUILD-HEAP | C: $O(n \lg n)$ |
| □ | Strassen's | D: $O(n^3)$ |
| □ | Randomized Quicksort | E: $O(n^2)$ |
| | | F: $O(n^{\lg 7})$ |

**Problem 2. Recurrences** (3 parts) [9 points]

Solve the following recurrences by giving tight Θ-notation bounds. You do not need to justify your answers, but any justification that you provide will help when assigning partial credit.

**(a)** $T(n) = T(\sqrt{n}) + \Theta(\lg\lg n)$

**(b)** $T(n) = T(n/2 + \sqrt{n}) + \sqrt{6046}$

**(c)** $T(n) = T(n/5) + T(4n/5) + \Theta(n)$

**Problem 3. Short Answers** (4 parts) [16 points]

Give *brief*, but complete, answers to the following questions.

**(a)** Argue that you cannot have a Priority Queue in the comparison model with both the following properties.

- EXTRACT-MIN runs in $\Theta(\lg \lg n)$ time.
- BUILD-HEAP runs in $\Theta(n)$ time.

**(b)** A sequence of $n$ operations is performed on a data structure. The $i$th operation costs $i$ if $i$ is a power of two, and one otherwise. Determine the amortized cost per operation.

**(c)** What does it mean to sort *in place*, and what is one advantage of sorting in place? Which of the following algorithms sort in place?

- INSERTION-SORT
- MERGE-SORT
- HEAPSORT
- COUNTING-SORT

**(d)** If an algorithm has running time $T(m) \leq 2^m$ for all $m$ which are powers of 2, and $T(n)$ is monotonically increasing, then can we conclude that $T(n) = O(2^n)$ by using the sloppiness lemma?

**(e)** Consider the following collection $\mathcal{H} = \{h_1, h_2, h_3\}$ of hash functions, where the three hash functions map the universe $\{A, B, C, D\}$ of keys into the range $\{0, 1, 2\}$ according to the following table:

| $x$ | $h_1(x)$ | $h_2(x)$ | $h_3(x)$ |
|---|---|---|---|
| $A$ | 1 | 0 | 2 |
| $B$ | 1 | 2 | 0 |
| $C$ | 2 | 2 | 2 |
| $D$ | 2 | 0 | 0 |

Is this collection of hash functions universal?

**Problem 4. True or False, and Justify** (7 parts) [28 points]

Circle **T** or **F** for each of the following statements to indicate whether the statement is true or false, respectively. If the statement is correct, briefly state why. If the statement is wrong, explain why. The more content you provide in your justification, the higher your grade, but be brief. Your justification is worth more points than your true-or-false designation.

**T F** There exists a pivot selection algorithm such that quicksort on runs in $O(n \lg n)$ time in the worst case.

**T F** Let $f$ and $g$ be asymptotically nonnegative functions. Then, at least one relationship of $f(n) = O(g(n))$ and $g(n) = O(f(n))$ must always hold.

**T F** Suppose we use a hash function $h$ to hash $n$ distinct keys into an array $T$ of length $m$. Assuming simple uniform hashing, the expected number of colliding pairs is $\Omega((\log n)/m)$.

**T F** Suppose that an array contains $n$ numbers, each of which is $-1$, 0, or 1. Then, the array can be sorted in $O(n)$ time in the worst case.

**T F** Suppose that a hash table of $m$ slots contains a single element with key $k$ and the rest of the slots are empty. Suppose further that we search $r$ times in the table for various other keys not equal to $k$. Assuming simple uniform hashing, the probability is $r/m$ that at least one of the $r$ searches probes the slot containing the single element stored in the table.

**T F** On all input arrays consisting of more than a 1000 elements, QUICKSORT performs at most as many comparisons as INSERTION-SORT.

**T F** Bucket sort can be used to sort an arbitrary list of real numbers in $O(n)$ expected time.

**Problem 5. Sorting a partially-sorted array** (3 parts) [10 points]

In this problem, more efficient algorithms will be given more credit. Partial credit will be given for correct but inefficient algorithms.

Let $A_0$ be a numerical array of length $n$, originally sorted into ascending order. Assume that $k$ entries of $A_0$ are overwritten with new values, producing an array $A$. Furthermore assume you have an array $B$ containing $n$ boolean values, where $B[i]$ is true if $A[i]$ is one of the $k$ values that was overwritten, and false otherwise.

**(a)** Give a fast algorithm to sort $A$ into ascending order, with time complexity better than $O(nk)$. [5 points]

**(b)** Give the time complexity of your algorithm in big-O notation, as a function of $n$ and $k$. [3 points]

**(c)** Give the space complexity of your algorithm in big-O notation, as a function of $n$ and $k$. (Do not include the space required for $A$ and $B$.) [2 points]

**Problem 6. Tree Ancestors**

Suppose you are given a complete binary tree of height $h$ with $n = 2^h$ leaves, where each node and each leaf of this tree has an associated "value" $v$ (an arbitrary real number).

If $x$ is a leaf, we denote by $A(x)$ the set of ancestors of $x$ (including $x$ as one of its own ancestors). That is, $A(x)$ consists of $x$, $x$'s parent, grandparent, etc. up to the root of the tree.

Similarly, if $x$ and $y$ are distinct leaves we denote by $A(x, y)$ the ancestors of *either* $x$ or $y$. That is,

$$A(x, y) = A(x) \cup A(y) \ .$$

Define the function $f(x, y)$ to be the sum of the values of the nodes in $A(x, y)$.

Give an algorithm (pseudo-code not necessary) that efficiently finds two leaves $x_0$ and $y_0$ such that $f(x_0, y_0)$ is as large as possible. What is the running time of your algorithm?

SCRATCH PAPER — Please detach this page before handing in your exam.

SCRATCH PAPER — Please detach this page before handing in your exam.

*Introduction to Algorithms* March 4, 2008
Massachusetts Institute of Technology 6.046J/18.410J
Professors Manolis Kellis and Piotr Indyk Practice Quiz 1

# Practice Quiz 1

**Problem 1. Algorithms and running times** (5 parts) [5 points]

Match each algorithm below with the tightest asymptotic upper bound for its worst-case running time by inserting one of the letters A, B, . . ., E into the corresponding box. **Some running times may be used multiple times or not at all.** For sorting algorithms, $n$ is the number of input elements. For matrix algorithms, the input matrix has size $n \times n$.

You need not justify your answers. Because points will be deducted for wrong answers, do not guess unless you are reasonably sure.

| | | |
|---|---|---|
| □ | Insertion sort | A: $O(\lg n)$ |
| □ | Binary Search | B: $O(n)$ |
| □ | BUILD-HEAP | C: $O(n \lg n)$ |
| □ | Strassen's | D: $O(n^3)$ |
| □ | Randomized Quicksort | E: $O(n^2)$ |
| | | F: $O(n^{\lg 7})$ |

**Solution:** From top to bottom: E, A, B, F, E.

**Problem 2. Recurrences** (3 parts) [9 points]

Solve the following recurrences by giving tight Θ-notation bounds. You do not need to justify your answers, but any justification that you provide will help when assigning partial credit.

**(a)** $T(n) = T(\sqrt{n}) + \Theta(\lg \lg n)$

**Solution:** We start by using a change of variable, $n = 2^m$, and we define $S(m) = T(2^m)$. This translates the recurrence into $S(m) = S(m/2) + \Theta(\lg m)$. By the master method, $S(m) = \Theta((\lg m)^2)$. By changing the variable nack to $n$, we obtain $T(n) = \Theta((\lg \lg n)^2)$.

**(b)** $T(n) = T(n/2 + \sqrt{n}) + \sqrt{6046}$

**Solution:** We neglect the lower order term $\sqrt{n}$ and we use the master method to obtain $T(n) = \Theta(\lg n)$. We may use the substitution method to verify this answer.

**(c)** $T(n) = T(n/5) + T(4n/5) + \Theta(n)$

**Solution:** We use a recursion tree. The sum of the terms at each level is equal to $n$ and the height of the tree is $\lg n$. This gives $T(n) = \Theta(n \lg n)$.

**Problem 3. Short Answers** (4 parts) [16 points]

Give *brief*, but complete, answers to the following questions.

**(a)** Argue that you cannot have a Priority Queue in the comparison model with both the following properties.

- EXTRACT-MIN runs in $\Theta(\lg\lg n)$ time.
- BUILD-HEAP runs in $\Theta(n)$ time.

**Solution:** If such a priority queue exists, then, we can sort in $\Theta(n\lg\lg n)$ in the comparison model by using BUILD-HEAP and applying EXTRACT-MIN $n$ times. This contradicts the lower bound on the running time of sorting algorithms in the comparison model.

**(b)** A sequence of $n$ operations is performed on a data structure. The $i$th operation costs $i$ if $i$ is a power of two, and one otherwise. Determine the amortized cost per operation.

**Solution:** Over the course of $n$ operations, we have at most $\lceil \lg n \rceil$ operations that cost $i$ and at most $n - \lfloor \lg n \rfloor = \Theta(n)$ operations that cost 1. Thus, we have a total cost over $n$ operations of
$\Theta(n) + \sum_{i=0}^{\lceil \lg n \rceil} 2^i = \Theta(n) + (\frac{2^{\lceil \lg n \rceil - 1}}{2-1}) = \Theta(n)$
Because we are averaging over $n$ operations, we have an amortized cost of $\Theta(n)/n = \Theta(1)$.

**(c)** What does it mean to sort *in place*, and what is one advantage of sorting in place? Which of the following algorithms sort in place?

- INSERTION-SORT
- MERGE-SORT
- HEAPSORT
- COUNTING-SORT

**Solution:** An in place sorting algorithm does not use auxiliary array and sorts items by moving them around in the same array. One advantage of in place sorting is that it uses less memory, and has better cache performance. INSERTION-SORT and HEAPSORT work in place, while MERGE-SORT and COUNTING-SORT do not.

**(d)** If an algorithm has running time $T(m) \leq 2^m$ for all $m$ which are powers of 2, and $T(n)$ is monotonically increasing, then can we conclude that $T(n) = O(2^n)$ by using the sloppiness lemma?

**Solution:** Since $f(n) = 2^n$ is an exponential function, it does not grow slowly, and therefore sloppiness lemma does not apply.

**(e)** Consider the following collection $\mathcal{H} = \{h_1, h_2, h_3\}$ of hash functions, where the three hash functions map the universe $\{A, B, C, D\}$ of keys into the range $\{0, 1, 2\}$ according to the following table:

| $x$ | $h_1(x)$ | $h_2(x)$ | $h_3(x)$ |
|---|---|---|---|
| $A$ | 1 | 0 | 2 |
| $B$ | 1 | 2 | 0 |
| $C$ | 2 | 2 | 2 |
| $D$ | 2 | 0 | 0 |

Is this collection of hash functions universal?

**Solution:** Yes. A hash family $\mathcal{H}$ that maps a universe of keys $U$ into $m$ slots is *universal* if for each pair of distinct keys $x, y \in U$, the number of hash functions $h \in \mathcal{H}$ for which $h(x) = h(y)$ is exactly $|\mathcal{H}|/m$. In this problem, $|\mathcal{H}| = 3$ and $m = 3$. Therefore, for any pair of the four distinct keys, exactly 1 hash function should make them collide. By consulting the table above, we have:

| | | |
|---|---|---|
| $h(A) = h(B)$ | only for $h_1$ | mapping into slot 1 |
| $h(A) = h(C)$ | only for $h_3$ | mapping into slot 2 |
| $h(A) = h(D)$ | only for $h_2$ | mapping into slot 0 |
| $h(B) = h(C)$ | only for $h_2$ | mapping into slot 2 |
| $h(B) = h(D)$ | only for $h_3$ | mapping into slot 0 |
| $h(C) = h(D)$ | only for $h_1$ | mapping into slot 2 |

**Problem 4. True or False, and Justify** (7 parts) [28 points]

Circle **T** or **F** for each of the following statements to indicate whether the statement is true or false, respectively. If the statement is correct, briefly state why. If the statement is wrong, explain why. The more content you provide in your justification, the higher your grade, but be brief. Your justification is worth more points than your true-or-false designation.

**T F** There exists a pivot selection algorithm such that quicksort on $n$ numbers runs in $O(n \lg n)$ time in the worst case.

**Solution:** **True.** In $O(n)$ time we deterministically find the median and we use this median as the pivot.

**T F** Let $f$ and $g$ be asymptotically nonnegative functions. Then, at least one relationship of $f(n) = O(g(n))$ and $g(n) = O(f(n))$ must always hold.

**Solution:** **False.** For $f(n) = 1$ and $g(n) = \|n * \sin(n)\|$ it is false.

**T F** Suppose we use a hash function $h$ to hash $n$ distinct keys into an array $T$ of length $m$. Assuming simple uniform hashing, the expected number of colliding pairs is $\Omega((\log n)/m)$.

**T F** Suppose that an array contains $n$ numbers, each of which is $-1$, 0, or 1. Then, the array can be sorted in $O(n)$ time in the worst case.

**Solution:** **True.** We may use counting sort. We first add 1 to each of the elements in the input array such that the precondition of counting sort is satisfied. After running counting sort, we subtract 1 from each of the elements in the sorted output array.

A solution based on partitioning is as follows. Let $A[1 \,..\, n]$ be the input array. We define the invariant

- $A[1 \,..\, i]$ contains only $-1$,
- $A[i + 1 \,..\, j]$ contains only 0, and
- $A[h \,..\, n]$ contains only $+1$.

Initially, $i = 0$, $j = 0$, and $h = n + 1$. If $h = j + 1$, then we are done; the array is sorted. In the loop we examine $A[j+1]$. If $A[j+1] = -1$, then we exchange $A[j+1]$ and $A[i+1]$ and we increase both $i$ and $j$ with 1 (as in partition in quicksort). If $A[j + 1] = 0$, then we increase $j$ with 1. Finally, if $A[j + 1] = +1$, then we exchange $A[j + 1]$ and $A[h - 1]$ and we decrease $h$ by 1.

**T F** Suppose that a hash table of $m$ slots contains a single element with key $k$ and the rest of the slots are empty. Suppose further that we search $r$ times in the table for various other keys not equal to $k$. Assuming simple uniform hashing, the probability is $r/m$ that at least one of the $r$ searches probes the slot containing the single element stored in the table.

**Solution:** **False.** The probability $p$ that one of the $r$ searches collides with the single element stored in the table is equal to 1 minus the probability that none of the $r$ searches collides with the single element stored in the table. That is, $p = 1 - (1 - 1/m)^r$.

**T F** On all input arrays consisting of more than a 1000 elements, QUICKSORT performs at most as many comparisons as INSERTION-SORT.

**Solution:** **False.** If the input is already sorted, then QUICKSORT performs $n \lg n$ comparisons and insertion sort only needs $n$ comparisons.

**T F** Bucket sort can be used to sort an arbitrary list of real numbers in $O(n)$ expected time.

**Solution:** **False.** Violates the lower bound on comparison sorting. Bucket sort assumes a distribution on the inputs. It is not a linear time algorithm for arbitrary lists of real numbers since all numbers can be chosen to fall in the same bucket.

**Problem 5. Sorting a partially-sorted array** (3 parts) [10 points]

In this problem, more efficient algorithms will be given more credit. Partial credit will be given for correct but inefficient algorithms.

Let $A_0$ be a numerical array of length $n$, originally sorted into ascending order. Assume that $k$ entries of $A_0$ are overwritten with new values, producing an array $A$. Furthermore assume you have an array $B$ containing $n$ boolean values, where $B[i]$ is true if $A[i]$ is one of the $k$ values that was overwritten, and false otherwise.

(a) Give a fast algorithm to sort $A$ into ascending order, with time complexity better than $O(nk)$. [5 points]

**Solution:** A straightforward solution is: (i) Separate out $A$ into two lists, $A_1$ consisting of all elements of A where the corresponding element of B is false, and $A_2$ where the corresponding element is true. (ii) Sort $A_2$ using mergesort or heapsort. (iii) Perform a linear merge of $A_1$ and $A_2$, writing the result back into $A$.

Partial credit was given for solutions based on insertion sort.

**(b)** Give the time complexity of your algorithm in big-O notation, as a function of $n$ and $k$. [3 points]

**Solution:** Separation of lists: $O(n)$. Sorting new items: $O(k \lg k)$. Merging back together again: $O(n)$. Total time: $O(n + k \lg k)$. Note that if the algorithm given is correct, then any correctly-demonstrated time bound less than $O(nk)$ for the algorithm may be given here.

**(c)** Give the space complexity of your algorithm in big-O notation, as a function of $n$ and $k$. (Do not include the space required for $A$ and $B$.) [2 points]

**Solution:** $O(n)$. There are more efficient implementations that overwrite parts of $A$ and $B$ as they go, but some of these approaches sacrifice time efficiency for space efficiency, and optimization of space was not asked for in this question. As long as the algorithm is correct and runs in time less than $O(nk)$, the correct space complexity for the algorithm is all that is required here.

**Problem 6. Tree Ancestors**

Suppose you are given a complete binary tree of height $h$ with $n = 2^h$ leaves, where each node and each leaf of this tree has an associated "value" $v$ (an arbitrary real number).

If $x$ is a leaf, we denote by $A(x)$ the set of ancestors of $x$ (including $x$ as one of its own ancestors). That is, $A(x)$ consists of $x$, $x$'s parent, grandparent, etc. up to the root of the tree.

Similarly, if $x$ and $y$ are distinct leaves we denote by $A(x, y)$ the ancestors of *either* $x$ or $y$. That is,

$$A(x, y) = A(x) \cup A(y) .$$

Define the function $f(x, y)$ to be the sum of the values of the nodes in $A(x, y)$.

Give an algorithm (pseudo-code not necessary) that efficiently finds two leaves $x_0$ and $y_0$ such that $f(x_0, y_0)$ is as large as possible. What is the running time of your algorithm?

**Solution:** There are several different styles of solution to this problem. Since we studied divide-and-conquer algorithms in class, we just give a divide-and-conquer solution here. There were also several different quality algorithms, running in $O(n)$, $O(n \lg n)$, and $O(n^2 \lg n)$. These were worth up to 11, 9, and 4 points, respectively. A correct analysis is worth up to 4 points.

First, let us look at an $O(n \lg n)$ solution then show how to make it $O(n)$. For simplicity, the solution given here just finds the maximum value, but it is not any harder to return the leaves giving this value as well.

We define a recursive function MAX1$(z)$ to return the maximum value of $f(x)$—the sum of the ancestors of a single node—over all leaves $x$ in $z$'s subtree. Similarly, we define MAX2$(z)$ to be a function returning the maximum value of $f(x, y)$ over all pairs of leaves $x, y$ in $z$'s subtree. Calling MAX2 on the root will return the answer to the problem.

First, let us implement MAX1$(z)$. The maximum path can either be in $z$'s left subtree or $z$'s right subtree, so we end up with a straightforward divide and conquer algorithm given as:

```
MAX1(z)
return ($value(z) + \max\{$MAX1($left[z]$), MAX1($right[z]$)$\}$)
```

For MAX2$(z)$, we note that there are three possible types of solutions: the two leaves are in $z$'s left subtree, the two leaves are in $z$'s right subtree, or one leaf is in each subtree. We have the following pseudocode:

```
MAX2(z)
return ($value(z) + \max\{$MAX2($left[z]$), MAX2($right[z]$), MAX1($left[z]$) + MAX1($right[z]$)$\}$)
```

**Analysis:**
For MAX1, we have the following recurrence

$$\begin{aligned} T_1(n) &= 2T_1\left(\frac{n-1}{2}\right) + \Theta(1) \\ &= \Theta(n) \end{aligned}$$

by applying the Master Method.

For MAX2, we have

$$\begin{aligned} T_2(n) &= 2T_2\left(\frac{n-1}{2}\right) + 2T_1\left(\frac{n-1}{2}\right) + \Theta(1) \\ &= 2T_2\left(\frac{n-1}{2}\right) + \Theta(n) \\ &= \Theta(n \lg n) \end{aligned}$$

by case 2 of the Master Method.

To get an $O(n)$ solution, we just define a single function, MAXBOTH, that returns a pair—the answer to MAX1 and the answer to MAX2. With this simple change, the recurrence is the same as MAX1

SCRATCH PAPER — Please detach this page before handing in your exam.

# 6.046 Exam 1

10/10/2012

$f(n) = O(g(n))$ asy upper

$0 \le f(n) \le c\,g(n)$

$f(n) = o(g(n))$ upper

$0 \le f(n) < c\,g(n)$

$f(n) = \Omega(g(n))$ asy lower

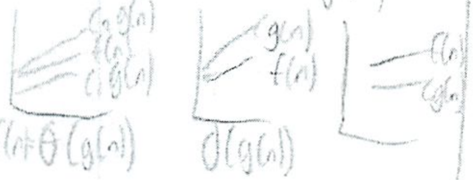
$0 \le c\,g(n) \le f(n)$

$f(n) = \omega(g(n))$ lower

$0 \le c\,g(n) < f(n)$

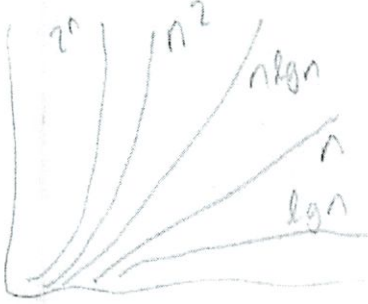
$f(n) = \Theta(g(n))$ asy tight = upper + lower

$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$



$f(n)=\Theta(g(n))$ $O(g(n))$

| | upper | both | lower |
|---|---|---|---|
| = | → | $\Theta$ | ← |
| ≤ | $O$ | ∧ | $\Omega$ |
| < | $o$ | × | $\omega$ |



## Interval Scheduling

Greedy

weighted : DP

$\max(w_i + \text{Opt}(R^f(i))$

Diff servers : impossible

Monte Carlo small prob incorrect
deterministic run time

Las Vegas always correct
random makes faster

## Convex Hull

Divide + Conquer

## Median Finding

Cols each w/ 5 items each
median of that
to get x ⇒ the center
which we can recurse
right + left

## Skip Lists

randomized data structure

```
14 ------> 50 ----> 79
14 --> 34 --> 50 --> 66 --> 79
14 23 34 42 50 59 66 72 79
```

EV = weighted avg of all possibilities

Freivalds $A \cdot B = C$ matrix
get random rs from {0,1}
some good, some bad

## FFT

Convert from coeff to pt value form
then can add or multiply $O(n)$
but this conversion must be efficient
choose $\pm x_0, \pm x_1, \ldots \pm x_{\frac{n-1}{2}}$

$A(x_i) = a_0 + a_1 x_i + a_2 x_i^2 + \ldots$

$A(-x_i) = a_0 - a_1 x_i + a_2 x_i^2 + \ldots$

$A_{even} = a_0 + a_2 x + a_4 x^2 + \ldots$

$A_{odd} = a_1 + a_3 x + a_5 x^2 + \ldots$

$A(x_i) = A_{even}(x_i^2) + x_i A_{odd}(x_i^2)$

$A(-x_i) = A_{even}(x_i^2) - x_i A_{odd}(x_i^2)$

reduce to 2 subproblems, each deg $< \frac{n}{2}$

Better notation

$e^{iu} = \cos(u) + i\sin(u)$

Sols to $w^n = 1$ are $e^{2\pi i k/n}$ for $k = 0, \ldots n-1$

```
Alg FFT(A, n)
    if n=1, return A(1)
    write A(x) as
        A(x) = Aeven(x^2) + Aodd(x^2)
    Call FFT(Aeven, n/2)
    Call FFT(Aodd, n/2)
    Compute A at n powers of w_n
        A(w_n^j) = Aeven(w_n^{2j}) + w_j Aodd(w_n^{2j})
    return A(w_n^0) A(w_n^1) ...
```

$w_n = e^{2\pi i/n}$

$w_{n_j} = -w^{n/2+j} = e^{\frac{2\pi i(n/2+j)}{n}}$
$= (e^{2\pi i/2})(e^{2\pi i j/n})$
$= -w_n^{n/2+j}$

$$\begin{bmatrix} A(x_0) \\ A(x_1) \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^1 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & & x_1^{n-1} \\ \vdots & & \ddots & & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

Value rep

Vandermode matrix $M$

Coefficient rep

**Eval** multiply by $M$

**intep** multiply by $M^{-1}$

## Polynomial identies

Given inputs that are distinct for each possibly $y$, exactly one degree $\leq n$ polynomial

$f(x_0) = y_0$

$f(x_1) = y_1$

$f(x_n) = y_n$

eg 2 pts detemine a line
3 pts quadratic polynomial

So randomly try evaluating till disprove!

## DP

1. Characterize optimal soln
2. Recursivly define the value of an optimal sol from optimal subproblems
3. Compute value of optimal in bottom up fashion

top down: recurse + memoize

bottom up: Iterative

What is subproblem?

Memoize!

## Master Theom

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$\frac{n}{b}$ = # of recursive calls

Look at $n^{\lg_b a}$ vs $f(n)$

For each $n^{c}(\lg n)^d$

1. Compare $c$s
   - bigger $c$
2. Compare $d$s
   - bigger or $\lg n$

$n^{\lg_b a} \cdot \lg n$

## Single Source Shortest Path

| | | |
|---|---|---|
| Unweighted | BFS | $O(n+m)$ |
| non neg weights | Dij | $O(m+n\log n)$ |
| general | Bellman Fd | $O(nm)$ |
| acyclic | Topo sort + 1 pass B.F. | $O(n+m)$ |

**topo sort** DFS, sort by finishing time $O(V+E)$

**strongly connected** from one point to any other

**Min Spanning Tree** $A$ is min spanning tree

greedy: find an edge so $A$ still ? repeat, return $A$

```
MST-Kruskal(G, w)
    A = ∅
    for each vertex $v \in G.V$
        Make-Set(v)
    Sort the edges G.E in nondec order by weight
    for each edge $(u,v) \in G.E$
        if FindSet(u) ≠ FindSet(v)
            $A = A \cup \{(u,v)\}$
            Union(u, v)
    return A
```

$O(E \lg V)$

```
MST-Prim(G, w, r)
    for each $u \in G.V$
        $u.key = \infty$
        $u.\pi = nil$
    $r.key = 0$
    $Q = G.V$
    while $Q \neq \emptyset$
        $u = $ Extract-Min(Q)      ← based on key
        for each $v \in G.adj[u]$
            if $v \in Q$ and $w(u,v) < key$
                $v.\pi = u$
                $v.key = w(u,v)$
```

```
Floyd Warshall(W)
    n = W.rows
    $D^{(0)} = W$
    for k = 1 to n
        let $D^{(k)} = d_{ij}^{(k)}$ be new n×n matrix
        for i 1 to n
            for j 1 to n
                $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
```

## Johnson $O(V^2 \lg V + VE)$

1. Find $h$ s.t. $w_h(u,v) \geq 0$
   Set $h(v) = \delta(s,v)$ w/ Bellman Fd
2. Reweight all edges via
   $w_h(u,v) = w(u,v) + h(u) - h(v)$
3. Run Dij for all source nodes $u \in V$ using $w_h$
4. Reweight all edges $w(u,v) \in w_h(u,v) - h(u) + h(v)$

## Network Flow

Residual $c_f(u,v) = c(u,v) - f(u,v) > 0$

10/11

*Design and Analysis of Algorithms* October 11, 2012
Massachusetts Institute of Technology 6.046J/18.410J
Profs. Srini Devadas and Ronitt Rubinfeld Quiz 1

# Quiz 1

- Do not open this quiz booklet until you are directed to do so. Read all the instructions first.
- The quiz contains 5 problems, several with multiple parts. You have 80 minutes to earn 80 points.
- This quiz booklet contains 13 pages, including this one, and a sheet of scratch paper.
- This quiz is closed book. You may use one double-sided letter ($8\frac{1}{2}'' \times 11''$) or A4 crib sheet. No calculators or programmable devices are permitted. Cell phones must be put away.
- Write your solutions in the space provided. If you run out of space, continue your answer on the back of the same sheet and make a notation on the front of the sheet.
- Do not waste time deriving facts that we have studied. Just cite results from class.
- When we ask you to "give an algorithm" in this quiz, describe your algorithm in English or pseudocode, and provide a short argument for correctness and running time. You do not need to provide a diagram or example unless it helps make your explanation clearer.
- Do not spend too much time on any one problem. Generally, a problem's point value is an indication of how many minutes to spend on it.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Please be neat.
- Good luck!

| Problem | Title | Points | Parts | Grade | Initials |
|---|---|---|---|---|---|
| 0 | Name | 1 | 1 | 1 | |
| 1 | True or False | 20 | 10 | 8 | |
| 2 | Short Answers | 24 | 6 | 4 | |
| 3 | Discovering Fakes | 10 | 2 | 6 | KF |
| 4 | Finding Repetitions | 10 | 2 | 6 | SD |
| 5 | Dynamic Quiz Takers | 15 | 3 | 7 | AC |
| Total | | 80 | | 32 | |

3 | 3 | 2
2 | 2 | 0

Avg = 49

Name: Michael Plasmeier

Circle your recitation:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| F10 | F11 | [✓ F11] | F12 | F12 | F1 | F2 | F3 |
| R01 | R02 | [✓ R07] | R03 | R08 | R04 | R05 | R06 |
| Yotam | Boon Teik | [✓ Aizana] | Annie | Aizana | Annie | Katherine | Heejung |

1st 35 min left

exam is much more on facts than pratice
would have put on cheat sheet.
Why no run time on cheat sheet?

**Problem 0. Name.** [1 points] Write your name on every page of this exam booklet! Don't forget the cover.

**Problem 1. True or False.** [20 points] (10 parts)

Circle **T** or **F** for each of the following statements to indicate whether the statement is true or false and briefly explain why.

**(a)** **T** [✓ **F**] Given $n$ numbers, the worst-case running time of the median finding algorithm is $\Theta(n \log n)$.

+3 $\Theta(n)$

We find x somewhat in middle w/ special trick
Then we split in half + recurse on each section
This is $n + \frac{n}{2} + \frac{n}{4} = O(n)$ ? groups of 5, pivot on median of medians

**(b)** [✓ **T**] [✓ **F**] On the same input on different executions, the multivariate polynomial identity testing algorithm (from recitation 3) may produce different answers.

0

Monte Carlo? ?

It will run w/ diff time lenghts - but is always correct
since it tries every input till it stops
L Las Vegas

**(c)** [✓ **T**] **F** In the worst case, the running time for a search in a skip list is $\Theta(n)$.

+2

Must search through every entry
a very bad, non special skip list, but one nevertheless technically

**(d)** [✓ **T**] [✓ **F**] Given a set of points $\{x_1, \ldots, x_n\}$, and a degree $n$ bounded polynomial $P(x) = a_0 + a_1x + \ldots + a_{n-1}x^{n-1}$, we can evaluate $P(x)$ at $\{x_1, \ldots, x_n\}$ in $O(n \log n)$ time using the FFT algorithm presented in class.

each pt individually would be $n^2$
- too slow

0

FFT method is faster

**(e)** T [✓ F] If we augment the "paranoid" quicksort (from lecture) to only pick each potential pivot once, the worst-case running time of "paranoid" quicksort is worse than that of the randomized quicksort.

+1

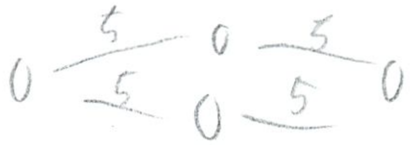
running time would be the same asy
as we are not saving that much work

**(f)** [✓ T] F In a weighted connected graph $G = (V, E)$, *each* edge with the minimum weight belongs to *some* minimum spanning tree of $G$.

? what min weight?
of all of them

+1

then worst case they are all the same



So not all edges would be in MST
but could they be? ∈ could be in some MST
depending on what random choices you make

**(g)** T [✓ F] In a weighted connected graph $G$, if $s$ is a starting node in Prim's algorithm, then for any other vertex $v$, the path on the resulting MST from $s$ to $v$ is the shortest path.

Shortest path ≠ min spanning tree
↳ Since this has diff goals
Connect each node at least once

+ 1

example?

**(h)** **T** [✓ **F**] If

$$T(n) = \begin{cases} T(an) + T(bn), & \text{if } n > n_0 \\ c, & \text{otherwise} \end{cases}$$

where $a, b > 0$ and $a + b < 1$, and $c$ and $n_0$ are some constants, then $T(n)$ is $O(n)$.

Say $T(\frac{1}{2}n) + T(\frac{1}{2}n)$

but that would be $n^2$

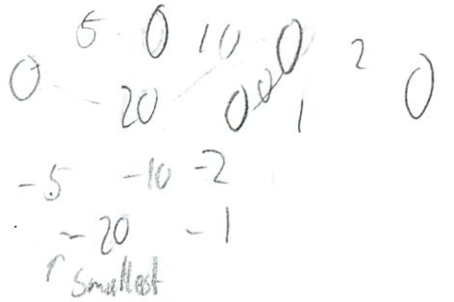
$$\frac{n}{\frac{n}{n}} \downarrow n$$

Since always doing both halves → n across n rows since we go till $n_0$ ) $O(n^2)$

**(i)** **T** [✓ **F**] For a given weighted connected graph $G = (V, E)$, we would like to find the longest simple path between any two vertices. We can solve this problem by negating the edge weights and running Johnson's algorithm. ? instead of FW

?, simple path = ?

?



✓

We would not take 0s which might bring us to somewhere longer

**(j)** [✓ **T**] **F** In a weighted graph $G$, if $k$ is the maximum number of edges in shortest paths between any two vertices, then it is possible to reduce the running time of Floyd-Warshall to $O(kn^2)$ by finishing early.

As saw in the HW - can reduce it

k is the # we go up to

$n^2$ since it does every i, j

Iterate over vertices

**Problem 2. Short Answers.** [24 points] (6 parts)

Please answer the following questions.

**(a)** Give asymptotic upper and lower bounds for $T(n)$ in the following recurrences. Make your bounds as tight as possible. You can assume that $T(n)$ is constant for $n \leq 2$.

$$T(n) = T(n^{1/3}) + T(n^{2/3}) + \log n$$

upper bound = time at each level · # of levels

n at each level - takes log n

never gets smaller ;

$O(n \lg n)$

lower bound as well     not tight

$\Theta(n \lg n)$     0

**(b)** We would like to test whether $A \times B = C$, where $A, B$, and $C$ are $n \times n$ matrices. Suppose it takes $O(k)$ to generate a random bit. Give an algorithm for the matrix identity test such that the error rate is less than $\epsilon$. What is the running time of your algorithm?

Frevidalds 'algorithm

Calculate random r from $\{0, 1\}$ → takes $O(n\,k)$

# cols n     time for each k

Test $Ar \cdot Br = Cr$

not all rs work

If don't match → return false

keep trying n till it works     X     2

$O(n^2 k)$

Said in lecture

Correction: Flows as a set of #s - over all - not 1 #

(c) Prove or disprove using a counterexample: Let $G$ be a flow network graph. If all the capacities of $G$ have unique integer capacities, then there is a unique maximum flow.

Proof by carter example.

Assume there existed a network flow that was not maximum

Then you could increase that to maximum

Assm network flow was maximum

Could not increase it futher

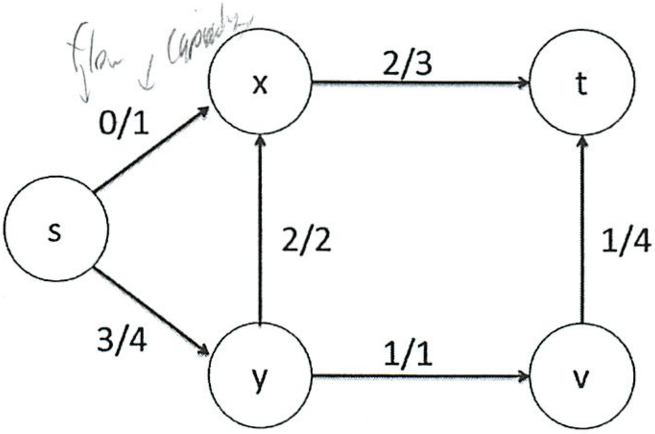
If non unique - could rearrange - but unique so we can't

But if max, can't increase it any more

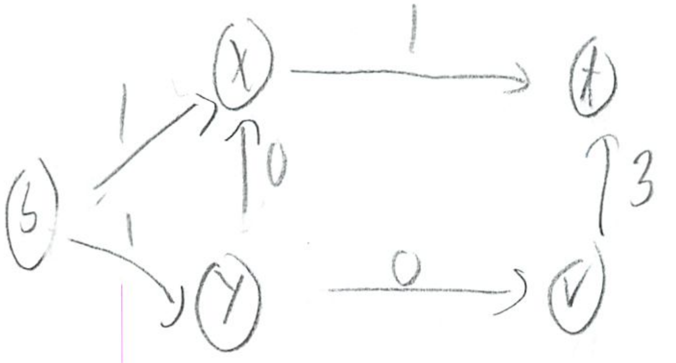
Thus contridiction, so unique capacites must lead to a unique max flow

-4

**(d)** Draw the residual graph $G_f$ given the graph $G$ and the flow $f$ below. The notation $x/y$ means that there is currently $x$ flow going through an edge of capacity $y$.



$G_f = C(u,v) - f(u,v)$



f is not maximum since path s→t could increase ~~a~~ flow

−2

backwards arrows?

**(e)** For a set $A$ of integers in the range $[0...100n]$, give an $O(n \log n)$ algorithm which finds whether there is a triple of three (not necessarily distinct) elements $(x, y, z)$ in $A$ such that $z = x + y + c$ where $c \in [-\log n, ..., +\log n]$. Justify the runtime.

Divide + Conquer

0

**(f)** You would like to design an algorithm which has the following specification: Given four polynomials $A, B, C, D$, each of degree at most $n$. If $AB = CD$, output PASS with probability 1. If $AB \neq CD$, output FAIL with probability at least $1 - 1/\log n$.

Your algorithm does the following *once*: chooses uniformly at random an integer $x$ from a set of integers $\{1, ..., m\}$ and checks if $A(x) \times B(x) = C(x) \times D(x)$. If the answer is yes, the algorithm outputs PASS, otherwise the algorithm outputs FAIL. How big should $m$ be in order to satisfy the specification?

That error $= 1 - \frac{1}{\log n}$

m should be up to log n

That would give it enough chanches to try something that it would fail in $1 \quad \frac{1}{\log n}$ times

0

have a feeling not what looking for

Name Michael Plasmeier

**Problem 3. Finding Fake Coins** [10 points] (2 parts) all real have same weight?

You are given a bag of coins, most of which have same weight. There is a possibility that some fake coins (which have different weight) are mixed into the bag. You want to find the fake coins or make sure that all coins in the bag are real using a scale that can compare the weight of a set of coins to another set of coins. (There is no way to measure absolute weights of coins.) Only rel

For the following settings, give the most efficient deterministic algorithm to find the fake coin(s) or show that there are none. Write a recurrence for your algorithm and solve the recurrence.

**(a)** A bag of $n$ coins may contain up to 1 fake coin that is heavier than real coins. Find the fake coin if it is in the pile.

3/5

Pick 2 random coins from bag → A, B $O(\frac{n}{2}) = O(n)$

Compare if =

If yes → discard both $O(\frac{n}{2})$

If no → try each one with another coin from bag → C

A C

B C $O(3)$

The one that is consistently diff is it

No recurrence but $O(\frac{n}{2}) \rightarrow O(n)$

→ over

**(b)** A bag of $n$ coins may contain up to 2 heavy fake coins. Find both fake coins in the pile if there are 2 or find one if there is 1.

3/5

Pick 4 random coins → A, B, C, D

A B vs C D

Then move them over, get a new A $O(3n)$

$E \curvearrowright A \curvearrowright B \curvearrowright C \curvearrowright D$ ← must retry D at end

E A vs B C

Similar check if consistently even

When ≠, start checking each combo 1 vs 1 $O(16)$

↳ must check 2 others (what was called C last time) to make sure not comparing both fakes

No recurrence

≠ O(n)

Alt ans to a w/ recurrence

You could also put half on scale A, half on scale B

And if they are different, investigate both halves

If the same, rotate half off of A onto B

half B → A

retry

But this has big chance need to rotate a lot to find

$2T(\frac{n}{2}) + O(n)$

$n \lg n$ ← worse!

too easy

**Problem 4. Finding Repetitions** [10 points] (2 parts)

**(a)** [4 points] Suppose there exists an integer that appears more than $n/2$ times in the array. Give a linear-time deterministic algorithm to find such an integer.

Nieve method: Track the freq that we have seen each # in a direct access table. (assumes bound on size of max int) or ∞ memory

So lookup + update = O(1) for n items

huge

3

= O(n)

Return when any count over $\frac{n}{2}$

**(b)** [6 points] Let $k$ be a given integer constant where $k \geq 2$. Now suppose there exists an integer that appears more than $n/k$ times (you can assume $n$ is divisible by $k$). Give a linear-time deterministic algorithm to find all such integers.

Similar. Store + update a freq table O(1)

When any freq > $\frac{n}{k}$, add to solutions

Do this for n items

Return list at end (since asks for all)

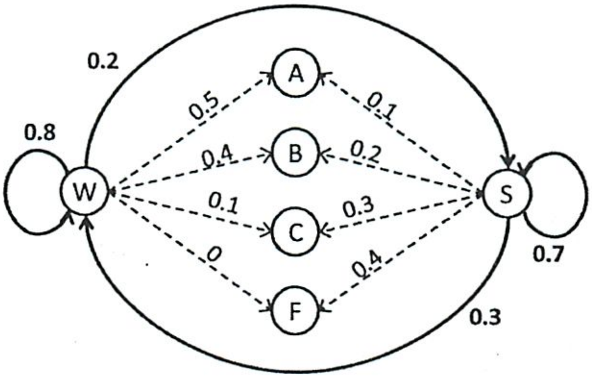
= $\Theta(n)$

3

**Problem 5. Dynamic Quiz Takers** [15 points] (3 parts)

In a fictional class, every student takes a stance in each quiz – either to work hard (W), or to slack off (S). 50% of the hard-workers receive an A for the quiz, 40% a B, 10% a C, while for slackers, only 10% receive an A, 20% a B, 30% a C, and 40% fail the quiz. Moreover, 20% of the hard-working students in one quiz become slackers in the next quiz, while 30% of the slackers in one quiz become hard-workers in the next quiz, independent of all previous quizzes. This model is summarized by the following diagram. The class begins with 80% hard-workers and 20% slackers, and has a series of $n$ quizzes.

markov chain



**(a)** [4 points] What is the proportion of students that receive $(A, F)$ for their first two quizzes, in order? Given a student with grade history $(A, F)$, what stances did the student most likely take (i.e., what is the stance history of the largest proportion, among students with such grade history)?

2

Fact: Given events $U, A, B, X$ such that $A \cup B = U$ and $A \cap B = \phi$,

$$\Pr(X|U) = \Pr(X|A)\Pr(A|U) + \Pr(X|B)\Pr(B|U).$$

$P((A,F)) = P(A|W)P(W) + P(A|S)P(S)$

1st round .5 .8 .1 .2

stay W / \ switch S stays S / \ switch W

$P(F|W \to W)P(W \to W) + P(F|S)P(W \to S)$ .4 .7 0 .3

0 .8 .4 $\boxed{.3}$ 0.2

$= \underline{.40}(0 + .12) + .02(.28 + 0)$

$= .048 + .0056$

$= .0536 = 5.3\%$

most popular was W ✓ switches to S

.048
.0056
.0536

.40
.12
80
.0400

.28
.02
.0056

**(b)** [8 points] Given the grade history of a student, $G_n = (g_1, ..., g_n)$, where $g_i \in \{A, B, C, F\}$ for $i = 1, ..., n$, we wish to determine the most probable stance history of the student, $H(G_n) = (h_1, ..., h_n)$, where $h_i \in \{W, S\}$ for $i = 1, ..., n$. 2

For example, if $n = 2$, given $G_2 = (A, A)$, the algorithm should return $H(G_2) = (W, W)$; given $G_2 = (A, F)$, the algorithm should return your answer in part (a).

Devise an algorithm that computes this using dynamic programming.

We define a recursion to do what I did on the previous page.

P(1st letter) = P(1st letter | W) P(W) + P(1st letter | S) P(S)

P(2nd letter given W or S) = P(2nd letter | W) P(W → W) + P(2nd letter | S) P(W → S)

= P(2nd letter | W) P(S → W) P(2nd letter | S) P(S → S)

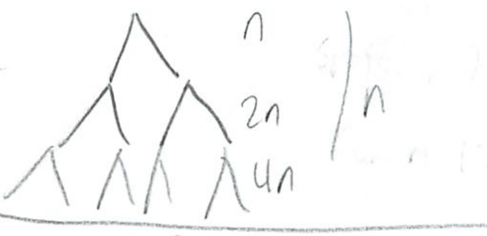
L = max( P(1st letter | W) P(W) + P(1st letter | S) P(S))

take max + recurse further, Since all # < 1 will only get smaller So take the max.

**(c)** [3 points] What is the runtime complexity of your algorithm, in terms of $n$? Compare this with the complexity of the brute force approach (i.e., the solution that iterates through all possible stance histories) to demonstrate the advantage of dynamic programming. 3

All stances / brute force

recursion tree



/ n O(n²)

DP; have some cut off - or else no difference

Only do one side → since # will only get smaller =O(n)

the larger side, still n levels O(n) but O(1) each level

SCRATCH PAPER

10/17

I totally misstudied for this
much more facts from lecture/notes
then come up on own
which is better - had I preped

Cheat sheet all wrong!

Coins 2

FF RR
RF RF

Move them across

vs

$O(3n) = O(n)$

It sounds like they want a recurrence

Divide + conquer

half half

would be $O(n \lg n)$ or $O(n)$

$2n(\frac{n}{2}) + O(1)$

which is mine

$n^{\log_2 2}$

$n$

②

rederived running time!

*Design and Analysis of Algorithms*
Massachusetts Institute of Technology
Profs. Srini Devadas and Ronitt Rubinfeld

October 11, 2012
6.046J/18.410J
Quiz 1 Solutions

# Quiz 1 Solutions

- Do not open this quiz booklet until you are directed to do so. Read all the instructions first.
- The quiz contains 5 problems, several with multiple parts. You have 80 minutes to earn 80 points.
- This quiz booklet contains 13 pages, including this one, and a sheet of scratch paper.
- This quiz is closed book. You may use one double-sided letter ($8\frac{1}{2}'' \times 11''$) or A4 crib sheet. No calculators or programmable devices are permitted. Cell phones must be put away.
- Write your solutions in the space provided. If you run out of space, continue your answer on the back of the same sheet and make a notation on the front of the sheet.
- Do not waste time deriving facts that we have studied. Just cite results from class.
- When we ask you to "give an algorithm" in this quiz, describe your algorithm in English or pseudocode, and provide a short argument for correctness and running time. You do not need to provide a diagram or example unless it helps make your explanation clearer.
- Do not spend too much time on any one problem. Generally, a problem's point value is an indication of how many minutes to spend on it.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Please be neat.
- Good luck!

| Problem | Title | Points | Parts | Grade | Initials |
|---|---|---|---|---|---|
| 0 | Name | 1 | 1 | | |
| 1 | True or False | 20 | 10 | | |
| 2 | Short Answers | 24 | 6 | | |
| 3 | Discovering Fakes | 10 | 2 | | |
| 4 | Finding Repetitions | 10 | 2 | | |
| 5 | Dynamic Quiz Takers | 15 | 3 | | |
| Total | | 80 | | | |

**Name:** ____________________

Circle your recitation:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| F10 | F11 | F11 | F12 | F12 | F1 | F2 | F3 |
| R01 | R02 | R07 | R03 | R08 | R04 | R05 | R06 |
| Yotam | Boon Teik | Aizana | Annie | Aizana | Annie | Katherine | Heejung |

**Problem 0. Name.** [1 points] Write your name on every page of this exam booklet! Don't forget the cover.

**Problem 1. True or False.** [20 points] (10 parts)

Circle **T** or **F** for each of the following statements to indicate whether the statement is true or false and briefly explain why.

**(a) T F** Given $n$ numbers, the worst-case running time of the median finding algorithm is $\Theta(n \log n)$.

**Solution:** [2 points] False. The running time of the median finding algorithm is $O(n)$.

**(b) T F** On the same input on different executions, the multivariate polynomial identity testing algorithm (from recitation 3) may produce different answers.

**Solution:** [2 points] True. If two polynomials are not identical, then it's possible to get both true and false answers when executing the test.

**(c) T F** In the worst case, the running time for a search in a skip list is $\Theta(n)$.

**Solution:** [2 points] True. $O(\log n)$ was expected, but in the worst case, we may have to search through $\Theta(n)$ elements. Recall that upon an insertion of an element, promotion of the element to the upper levels is done randomly. In the worst case, it's possible that no elements get promoted or all elements get promoted to the upper levels.

**(d) T F** Given a set of points $\{x_1, \ldots, x_n\}$, and a degree $n$ bounded polynomial $P(x) = a_0 + a_1 x + \ldots + a_{n-1}x^{n-1}$, we can evaluate $P(x)$ at $\{x_1, \ldots, x_n\}$ in $O(n \log n)$ time using the FFT algorithm presented in class.

**Solution:** [2 points] False. Using FFT, we can efficiently evaluate a polynomial at the roots of unity, but not any $n$ points.

**(e) T F** If we augment the "paranoid" quicksort (from lecture) to only pick each potential pivot once, the worst-case running time of "paranoid" quicksort is worse than that of the randomized quicksort.

**Solution:** [2 points] False. The worst-case running time of the randomized quicksort is $\Theta(n^2)$, and the augmented "paranoid" quicksort takes $O(n^2)$ time since each pivot takes $O(n)$ to check and there are $O(n)$ bad pivots, so the recursion for "paranoid" quicksort becomes $T(n) = T(n/4) + T(3n/4) + O(n^2)$. Using induction, one can show that $T(n) \leq cn^2$ for some constant $c$. Thus, it is not worse.

**(f) T F** In a weighted connected graph $G = (V, E)$, *each* edge with the minimum weight belongs to *some* minimum spanning tree of $G$.

**Solution:** [2 points] True. Consider a MST $T$. If it doesn't contain the minimum edge $e$, then by adding $e$ to $T$, we get a cycle. By removing a different edge than $e$ from the cycle, we get a spanning tree $T'$ whose total weight is no more than the weight of $T$. Thus, $T'$ is a MST that contains $e$.

**(g) T F** In a weighted connected graph $G$, if $s$ is a starting node in Prim's algorithm, then for any other vertex $v$, the path on the resulting MST from $s$ to $v$ is the shortest path.

**Solution:** [2 points] False. Consider graph $G$ with vertices $\{s, v, w\}$ and edge weights $e(s, v) = 3$, $e(s, w) = 2$ and $e(w, v) = 2$. Path on the MST from $s$ to $v$ is of cost 4, while the shortest path is 3. There are many examples possible.

**(h) T F** If

$$T(n) = \begin{cases} T(an) + T(bn), & \text{if } n > n_0 \\ c, & \text{otherwise} \end{cases}$$

where $a, b > 0$ and $a + b < 1$, and $c$ and $n_0$ are some constants, then $T(n)$ is $O(n)$.

**Solution:** [2 points] True. Can be proven by simple induction.

**(i) T F** For a given weighted connected graph $G = (V, E)$, we would like to find the longest simple path between any two vertices. We can solve this problem by negating the edge weights and running Johnson's algorithm.

**Solution:** [2 points] False. Negation of edge weights may create a negative cycle.

**(j) T F** In a weighted graph $G$, if $k$ is the maximum number of edges in shortest paths between any two vertices, then it is possible to reduce the running time of Floyd-Warshall to $O(kn^2)$ by finishing early.

**Solution:** [2 points] False. Floyd-Warshall iterates over a ordered list of vertices $(v_1, \ldots, v_n)$, and if the shortest path with length $k$ or less between two vertices uses node $v_n$, it will not find the shortest path until the last loop.

**Problem 2. Short Answers.** [24 points] (6 parts)

Please answer the following questions.

**(a)** Give asymptotic upper and lower bounds for $T(n)$ in the following recurrences. Make your bounds as tight as possible. You can assume that $T(n)$ is constant for $n \leq 2$.

$$T(n) = T(n^{1/3}) + T(n^{2/3}) + \log n$$

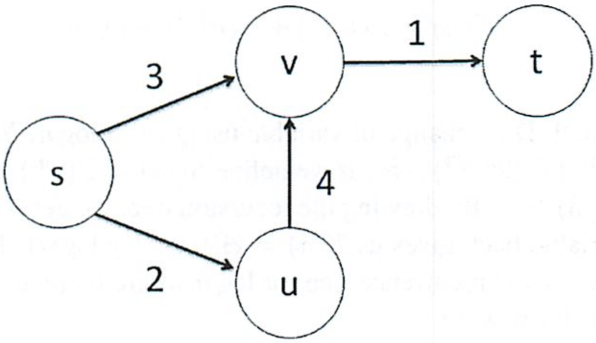
**Solution:** [4 points] Do a change of variable using $m = \log n$. You will end up with $T(2^m) = T(2^{m/3}) + T(2^{2m/3}) + m$. If we define $S(m) = T(2^m)$, we will get $S(m) = S(m/3) + S(2m/3) + m$. By drawing the recursion tree, we get $S(m) = \Theta(m \log m)$. Changing the variable back gives us $T(n) = \Theta(\log n \log \log n)$. It is also possible to solve this by drawing the recurrence tree for $\log n$ where there are $O(\log \log n)$ levels and each level has $\log n$ work

**(b)** We would like to test whether $A \times B = C$, where $A, B$, and $C$ are $n \times n$ matrices. Suppose it takes $O(k)$ to generate a random bit. Give an algorithm for the matrix identity test such that the error rate is less than $\epsilon$. What is the running time of your algorithm?

**Solution:** [4 points] If $A \cdot B \neq C$, Freivald's Algorithm has an error rate of less than $1/2$. Otherwise, it is always correct. Hence, the error rate will be less than $2^{-x}$ if we run the algorithm for $x$ times and output "equal" if the algorithm returns "equal" every time. Setting $2^{-x} = \epsilon$, we get $x = \log \frac{1}{\epsilon}$. Each time we run, we need to generate $n$ random bits. Therefore, the total running time is $O((nk + n^2) \log \frac{1}{\epsilon}$.

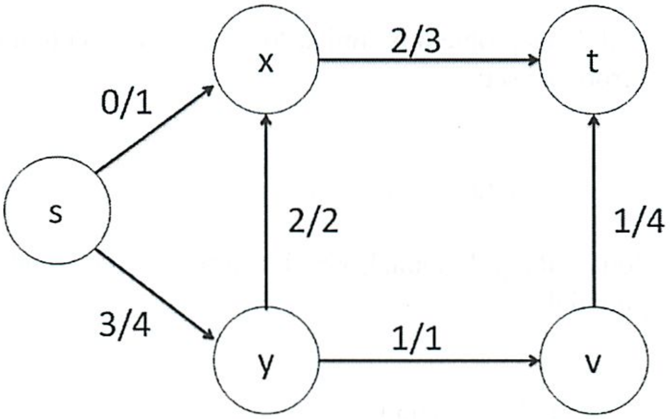
**(c)** Prove or disprove using a counterexample: Let $G$ be a flow network graph. If all the capacities of $G$ have unique integer capacities, then there is a unique maximum flow.

**Solution:** [4 points] This statement is false. Consider the graph given below:

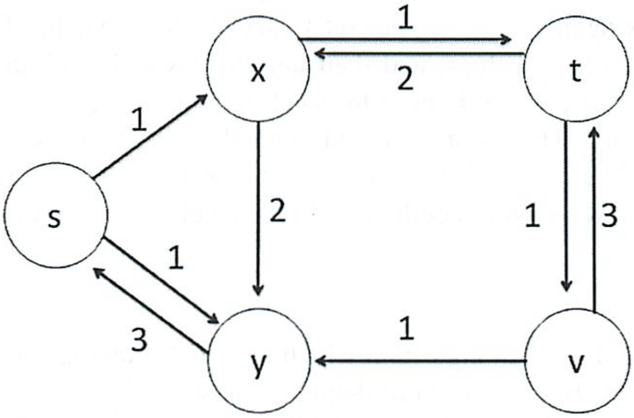


On this graph, the maximum flow is 1 because of the capacity on the edge $(v, t)$. However, it is possible to achieve this flow either by sending one unit through the path $(s, v, t)$ or to send one unit along $(s, u, v, t)$. Therefore, the flow is not unique even though the capacities all are.

**(d)** Draw the residual graph $G_f$ given the graph $G$ and the flow $f$ below. The notation $x/y$ means that there is currently $x$ flow going through an edge of capacity $y$.



**Solution:** [4 points] The residual graph is given below.



**(e)** For a set $A$ of integers in the range $[0...100n]$, give an $O(n \log n)$ algorithm which finds whether there is a triple of three (not necessarily distinct) elements $(x, y, z)$ in $A$ such that $z = x + y + c$ where $c \in [-\log n, ..., +\log n]$. Justify the runtime.

**Solution:** [4 points] This problem is similar to the problem on homework. Create a polynomial $P(x)$ from the set:

$$P(x) \quad = x^{a_1} + x^{a_2} + ...x^{a_{|A|}}$$

where $a_i \in A$. Square the polynomial, which can be done efficiently using FFT, to obtain a new polynomial

$$\begin{aligned} Q(x) &\equiv P(x)^2 \\ &= q_0 x^0 + q_1 x^1 + ... + q_{200n} x^{200n} \end{aligned}$$

Then $q_k$ will be exactly the number of pairs $i, j$ such that $a_i + a_j = k$ (note that $q_k$ might be zero). The first method then compares the first $100n$ resulting positive coefficient values against our original set to see if these match. This can be done by hashing the resulting values, and then iterating over the original set to test for inclusion. The same process is used to check for the $+i$ and $-i$ offset for all $i \in [-\log n, ..., +\log n]$. The second method notes that $Q(x)$ can be multiplied by the polynomial $x^{-\log n} + x^{-\log n+1} + \ldots + x^{-1} + 1 + x + \ldots + x^{\log n}$ using FFT and the powers of $x$ that have nonzero coefficients in the result can be compared to $A$ in time $O(n)$.

**(f)** You would like to design an algorithm which has the following specification: Given four polynomials $A, B, C, D$, each of degree at most $n$. If $AB = CD$, output PASS with probability 1. If $AB \neq CD$, output FAIL with probability at least $1 - 1/\log n$. Your algorithm does the following *once*: chooses uniformly at random an integer $x$ from a set of integers $\{1, ..., m\}$ and checks if $A(x) \times B(x) = C(x) \times D(x)$. If the answer is yes, the algorithm outputs PASS, otherwise the algorithm outputs FAIL. How big should $m$ be in order to satisfy the specification?

**Solution:** [4 points] $m$ should be $2n \log n$. If $AB = CD$, then the algorithm always outputs PASS. Otherwise, $AB \neq CD$. The product of two degree $\leq n$ polynomials has degree at most $2n$, so the probability that for random $x$, $A(x) \times B(x) = C(x) \times D(x)$ is at most $2n/m$ which by our choice of $m$ is at most $1/\log n$.

**Problem 3. Finding Fake Coins** [10 points] (2 parts)

You are given a bag of coins, most of which have same weight. There is a possibility that some fake coins (which have different weight) are mixed into the bag. You want to find the fake coins or make sure that all coins in the bag are real using a scale that can compare the weight of a set of coins to another set of coins. (There is no way to measure absolute weights of coins.)

For the following settings, give the most efficient deterministic algorithm to find the fake coin(s) or show that there are none. Write a recurrence for your algorithm and solve the recurrence.

**(a)** A bag of $n$ coins may contain up to 1 fake coin that is heavier than real coins. Find the fake coin if it is in the pile.

**Solution:** Divide the coins into 2 piles. Compare the two piles. If one pile is heavier, recurse on that pile. If they are the same, there is no fake coin.
$T(n) = T(\frac{n}{2}) + \Theta(1) = \Theta(\log n)$

**Alternative Solution:** Divide the coins into 3 piles. Choose two piles and compare them. If one pile is heavier, recurse on it. If they are even, the fake coin may be in the remaining pile or does not exist. Recurse on the pile you didn't compare.
$T(n) = T(\frac{n}{3}) + \Theta(1) = \Theta(\log n)$

**(b)** A bag of $n$ coins may contain up to 2 heavy fake coins. Find both fake coins in the pile if there are 2 or find one if there is 1.

**Solution:** [5 points] Divide the coins into 2 piles and compare them.
If the comparison is even, it means that (1) two piles each have one heavy coin or (2) there are no fake coins. Since there is at most one heavy coin in each pile, run algorithm from (a) on each pile to discover the fake coin in each pile. If the comparison is uneven, it means the heavy pile has all the fake coins. Recurse on the heavy pile.
Base case, you have two piles of one coin each. If one is heavier, it is a fake. If they are the same, you will have to test against a third coin.
The recursion is $T(n) = \begin{cases} T(\frac{n}{2}) + \Theta(1), & \text{if piles do not weigh the same} \\ \Theta(\log n), & \text{if piles weigh the same.} \end{cases}$.
Both cases have $\Theta(\log n)$ run time, so $T(n) = \Theta(\log n)$.

**Alternative Solution:** Divide the coins into 3 piles and compare them all there.
If they all weigh the same, then there are no fake coins. If two piles are heavier, combine these two piles and recurse on the combined pile. If one pile is heavier, there is one or two fake coins in it. Recurse on this pile. In the base case, you have 3 piles of 1 coin each. Weigh them against each other. The heavy two are the fake coins.
The recursion is $T(n) = T(\frac{2n}{3}) + \Theta(1) = \Theta(\log n)$.

**Problem 4. Finding Repetitions** [10 points] (2 parts)

**(a)** [4 points] Suppose there exists an integer that appears more than $n/2$ times in the array. Give a linear-time deterministic algorithm to find such an integer.

**Solution:** Use the deterministic linear-time median-finding algorithm covered in lecture to find the median of the array – it will be the majority element. Since, if the median wouldn't be the majority element, then the majority element would be smaller or larger than the median, which is impossible (e.g., there are only $n/2$ elements smaller than the median).

Note that hashing is not deterministic linear-time. Answers that used hashing received partial credit. One can imagine using an auxiliary array instead of a hash table and indexing into the array with the integers in the given array, but the size of this auxiliary array is unbounded.

**(b)** [6 points] Let $k$ be a given integer constant where $k > 2$. Now suppose there exists an integer that appears more than $n/k$ times (you can assume $n$ is divisible by $k$). Give a linear-time deterministic algorithm to find all such integers.

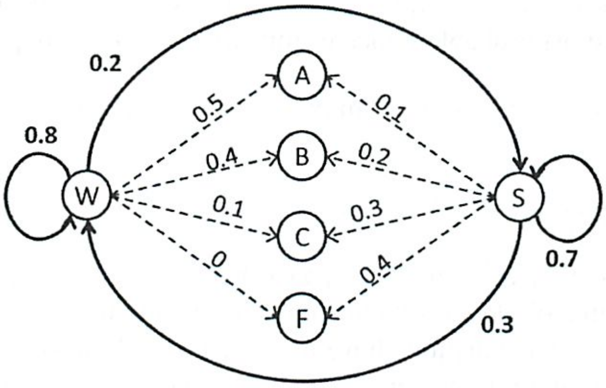
**Solution:** Let's call an integer that appears more than $n/k$ times a *common element.* Select elements $a_1, a_2, \ldots a_k$, which are the elements with ranks $\frac{n}{k}, \frac{2n}{k}, \ldots, \frac{(k-1)n}{k}, n$, respectively. The $k$ selections each takes $O(n)$ time. Then check each of the elements $a_1, a_2, \ldots a_k$, to see whether it is a common element. Each of the $k$ checks take $O(n)$ time. The overall runtime is therefore $O(nk)$.

The correctness of the algorithm follows from the claim that any common element must be among $a_1, a_2, \ldots a_k$. To see that, consider the sorted array. Any common element is a contiguous block of size $> n/k$. Since our "probes" $a_1, a_2, \ldots a_k$ are at distance $n/k$, they must strand the block of a common element.

See paragraph about hashing in the answer for part (a).

**Problem 5. Dynamic Quiz Takers** [15 points] (3 parts)

In a fictional class, every student takes a stance in each quiz – either to work hard (W), or to slack off (S). 50% of the hard-workers receive an A for the quiz, 40% a B, 10% a C, while for slackers, only 10% receive an A, 20% a B, 30% a C, and 40% fail the quiz. Moreover, 20% of the hard-working students in one quiz become slackers in the next quiz, while 30% of the slackers in one quiz become hard-workers in the next quiz, independent of all previous quizzes. This model is summarized by the following diagram. The class begins with 80% hard-workers and 20% slackers, and has a series of $n$ quizzes.



**(a)** [4 points] What is the proportion of students that receive $(A, F)$ for their first two quizzes, in order? Given a student with grade history $(A, F)$, what stances did the student most likely take (i.e., what is the stance history of the largest proportion, among students with such grade history)?

Fact: Given events $U, A, B, X$ such that $A \cup B = U$ and $A \cap B = \phi$,

$$\Pr(X|U) = \Pr(X|A)\Pr(A|U) + \Pr(X|B)\Pr(B|U).$$

**Solution:** 3.76%, $(W, S)$.

$$\begin{aligned}\Pr[(A,F)] &= \Pr[(A,F) \mid (W,S)] + \Pr[(A,F) \mid (S,S)] \\ &= \Pr[W \text{ for quiz } 1] \times \Pr[A|W] \times \Pr[S \text{ for quiz } 2 \mid W \text{ in quiz } 1] \times \Pr[F|S] \\ &\quad + \Pr[S \text{ for quiz } 1] \times \Pr[A|S] \times \Pr[S \text{ for quiz } 2 \mid S \text{ in quiz } 1] \times \Pr[F|S] \\ &= 0.8 \times 0.5 \times 0.2 \times 0.4 + 0.2 \times 0.1 \times 0.7 \times 0.4 \\ &= 0.032 + 0.0056 = 0.0376\end{aligned}$$

The proportion of students with grade history $(A, F)$ whose stances were $(W, S)$, which is $0.8 \times 0.5 \times 0.2 \times 0.4$, is larger than the proportion of students whose stances were $(S, S)$, which is $0.2 \times 0.1 \times 0.7 \times 0.4$. Stance histories $(W, W)$ and $(S, W)$ do not generate such a grade history. This means the students who received $(A, F)$ most likely had the stance history $(W, S)$.

**(b)** [8 points] Given the grade history of a student, $G_n = (g_1, ..., g_n)$, where $g_i \in \{A, B, C, F\}$ for $i = 1, ..., n$, we wish to determine the most probable stance history of the student, $H(G_n) = (h_1, ..., h_n)$, where $h_i \in \{W, S\}$ for $i = 1, ..., n$.

For example, if $n = 2$, given $G_2 = (A, A)$, the algorithm should return $H(G_2) = (W, W)$; given $G_2 = (A, F)$, the algorithm should return your answer in part (a).

Devise an algorithm that computes this using dynamic programming.

**Solution:** We memoize the proportion of the most probable stance history for each subproblem $G_k = (g_1, ..., g_k), k = 1, ..., n$. Let $V(G_k, h_k)$ denote the proportion of students in the most probable stance history with stance $h_k$ in quiz $k$. Then,

$$V(G_k, W) = \Pr[g_k|W] \times \max\{.8 \times V(G_{k-1}, W), .3 \times V(G_{k-1}, S)\}$$

and

$$V(G_k, S) = \Pr[g_k|S] \times \max\{.2 \times V(G_{k-1}, W), .7 \times V(G_{k-1}, S)\}.$$

The base cases, $V(g_1, W)$ and $V(g_1, S)$ where $g_1 \in \{A, B, C, F\}$, are given in the previous diagram. While constructing the table, record the maximum argument on the right-hand side of the equation above to allow for backtracking. The final solution is determined by finding the larger of $V(G_n, W)$ and $V(G_n, S)$, and then backtracking to give the most probable preceding stances.

**(c)** [3 points] What is the runtime complexity of your algorithm, in terms of $n$? Compare this with the complexity of the brute force approach (i.e., the solution that iterates through all possible stance histories) to demonstrate the advantage of dynamic programming.

**Solution:** The running time of the algorithm is $O(n)$, which is significantly more efficient than that of the brute-force solution, $O(2^n)$.